Traffic Classification for the Detection of Anonymous Web Proxy Routing

Shane Miller

School of Computing, Engineering & Intelligent Systems
Faculty of Computing & Engineering
Ulster University, Magee

A thesis submitted in partial fulfilment of the requirements for the degree of
Doctor of Philosophy

I confirm that the word count of this thesis is less than 100,000

Contents

Acknowledgements

I would like to take this opportunity to thank all of those who have, in some part, been involved with my PhD in one form or another. Firstly, I would like to give my sincerest thanks to my supervisors, Professor Kevin Curran and Dr Tom Lunney. Their guidance, support and dedication have been invaluable over the course of my research and have provided an excellent education in the many aspects of academic life.

I would like to extend many thanks to my colleagues and friends from Ulster University who provided so much support and experience over the years. The time spent with them discussing the various aspects of student and academic life made life easier and encouraged me through the challenges of the PhD. They also gave me a chance to break away from the rigors of research to relax and refresh myself

I would like to thank my family, in particular my brother Thomas, for supporting me through the entire process with unwavering belief in me. This belief has helped me achieve so much more than I expected and for that I'm eternally thankful.

Finally, I would like to dedicate this thesis my late mother Jennifer for believing in me and for constantly pushing me to aim higher and achieve what she never had the chance to.

Abstract

Network Proxies and Virtual Private Networks (VPN) are tools that are used every day to facilitate various business functions. However, they have gained popularity amongst unintended userbases as tools that can be used to hide mask identities while using websites and web-services. Anonymising Proxies and/or VPNs act as an intermediary between a user and a web server with a Proxy and/or VPN IP address taking the place of the user's IP address that is forwarded to the web server. For a business whose primary service is hosted on the internet, such as Facebook or Netflix, security systems are a vital part of these services; unauthorised user detection can be a vital feature of such systems. The detection of unauthorised users can be problematic for techniques that are available at present if the suspect users are using identity hiding tools such as anonymising proxies or VPNs.

This work presents computational models based on intelligent machine learning techniques to address the limitations currently experienced by unauthorised user detection systems. A model to detect usage of anonymising proxies was developed using a Multi-layered perceptron neural network that was trained using data found in the Transmission Control Protocol (TCP) header of captured network packets. Two models to detect usage of two different VPN configurations were also developed using a similar Multi-layered Perceptron neural network and were trained using flow statistics. The first model successfully classifies network traffic as either OpenVPN or as non-VPN traffic; the second model successfully classifies network traffic as either OpenVPN traffic that is tunnelled using Stunnel or as non-VPN traffic. Validation testing showed that the presented models are capable of classifying network traffic in a binary manner as direct (originating directly from a user's own device) or indirect (makes use of identity and location hiding features of proxies or VPNs) with high degrees of accuracy.

The proxy detection model additionally showed strong generalisation abilities when tested against multiple types of web-based anonymising proxies. These results demonstrate a significant advancement in the detection of unauthorised user access with evidence showing that there could be further advances for research in this field particularly in the application of business security.

Abbreviations

ACL – Access Control List

ANN – Artificial Neural Network

ARFF – Attribute-Relation File Format

AUC – Area Under Curve

CA – Certificate Authority

CDN – Content Distribution

CGI – Common Gateway Interface

CNN – Convolutional Neural Network

CSV – Comma Separated Value

DDoS – Distributed Denial of Service

DGSOT – Dynamically Growing Self-Organising Tree

DH – Diffie-Hellman

DNN – Deep Neural Network

DPI – Deep Packet Inspection

EAP – Extensible Authentication Protocol

EC2 – Elastic Compute Cloud

GRE – Generic Routing Encapsulation

HMAC – Hash-based Message Authentication Codes

HTTP – Hypertext Transfer Protocol

HTTPS – Hypertext Transfer Protocol

ICN – Information-Centric Network

IKE – Internet Key Exchange

IKEv2- Internet Key Exchange Version 2

IP – Internet Protocol

IPSec – Internet Protocol Security

ISAKMP – Internet Security Association and Key Management Protocol

ISP – Internet Service Provider

KNN – *k*-Nearest Neighbour

L2F – Layer 2 Forwarding

L2TP – Layer 2 Tunnelling Protocol

L2TPv3 – Layer 2 Tunnelling Protocol Version 3

LOOCV – Leave One Out Cross Validation

MiTM – Man in The Middle

MLaaS – Machine Learning as a Service

MS-CHAP – Microsoft Challenge-Handshake Authentication Protocol

NBA – Network Based Analysis

PAP – Password Authentication Protocol

PPP – Point-to-Point Protocol

PPTP – Point-to-Point Tunnelling Protocol

PRF – Pseudorandom Function

RNN – Recurrent Neural Network

SOCKS – Socket Secure

SPI – Shallow Packet Inspection

SSL – Secure Socket Layer

SSTP – Secure Socket Tunnelling Protocol

SVM – Support Vector Machine

TCP – Transmission Control Protocol

TTL – Time To Live

UDP – User Datagram Protocol

VM – Virtual Machine

VPN – Virtual Private Network

VPS – Virtual Private Server

WEKA – Waikato Environment for Knowledge Analysis

XSS – Cross Site Scripting

List of Figures

List of Tables

1. Introduction

The Internet has become an important part of everyday life and its usage continues to grow as more devices are released that have Internet connectivity. Internet usage in developing countries is especially increasing with the arrival of affordable mobile smartphones (Poushter, 2016). As more people use the Internet, governments seek to implement controls on what their citizens can access, either for the protection of said citizens against malware and identity theft or to suppress unacceptable parts of the Internet (King et al., 2017; Fiaschi et al., 2017; Gebhart & Kohno, 2017; Akabogu, 2017; Tanash et al., 2017). This leads some people to become concerned for their privacy as they do not want their online activities documented. Due to this and other factors, usage of technologies designed to provide anonymity on the Internet has increased (Anderson et al., 2017).

Anonymity technologies allow users of the Internet access to a level of privacy that prevents the recording of information such as IP addresses, which could be used to aid in the identification of the users. Users of these technologies will have varying motivations for why they want to protect their privacy. Some use anonymity technologies because they live in a country where their Internet usage is monitored and the websites that they wish to access are blocked. In this situation, the anonymity providing technology helps the user circumvent the blocks that have been imposed on them. A similar use case is a user preventing their browsing habits from being tracked by their Internet service provider (ISP). Some ISPs track browsing habits to improve the services that they provide while some collect the data so that it can be forwarded on to other third parties. These include advertisers who use it to produce targeted advertisements or possibly security forces who use it to build a profile of the suspects and determine whether they are adhering to a country's laws involving Internet access. Naturally, criminals want to avoid their identity being released to the police. Therefore, they turn to anonymity providing technologies. Anonymity systems transport network packets over intermediary relays so that no single system other than the original machine has information that could identify the user. Since many people can make use of these intermediary relays at the same time, the connection of the user seeking anonymity is hidden amongst the network traffic of other Internet users (Li et al., 2013). These different use-cases have led to anonymity on the Internet being a

divisive topic. On one side, anonymity technologies provide legitimate methods for protecting freedom of speech and privacy, facilitating the transfer of anonymous tips to law enforcement and bypassing state censorship. However, the same technologies can be used to provide protection to criminals who are involved in information and identity theft, spam emailing and even organised terrorism. Additionally, they can be used for network abuse by bypassing Internet usage policies of organisations. This has the potential to expose the internal workings of the organisation to malicious activities.

There are various types of anonymity technologies available with most being based on networks called “mix” networks. Mix networks use a chain of proxy servers to create communication pathways that are difficult to trace (Chaum, 1981). The anonymous communication systems that resulted from this can be categorised into one of two groups: message based/high-latency applications or flow based/low-latency applications (Yang et al., 2015). High latency applications can include email and e-voting systems. Low latency systems include the popular anonymous communication system Tor as well as various kinds of HTTP/SOCKS proxy services and Virtual Private Networks (VPNs) (Lee et al., 1996; Wood et al., 1988). Systems such as Tor fall under the category of multi-hop anonymous communications models, while HTTP/SOCKS proxies and VPNs generally fall under the category of single-hop anonymous communication models. The focus of this thesis will be on these single-hop anonymous communication models.

A proxy server is a server that acts as an intermediary for requests from clients for resources located on other servers on a network or the Internet. A basic type of proxy is a gateway which can be found on most consumer wireless routers. Another type of proxy is a reverse proxy which is a server on an internal company network that acts as an intermediary for other servers based on that network. Reverse proxies are typically used as an Internet facing server that handles several different tasks, load balancing being one of them. The proxy server distributes requests between several web servers and acts as a cache for static content such as pictures and other graphical content. Proxy servers that are used to provide anonymisation are based on another type of proxy known as an “open” proxy. Open proxies are a proxy that is available to any

user on the Internet. They are mostly used to set up anonymous proxy websites and categorised as a single-hop anonymous communication model.

There are several different implementations of VPNs for providing anonymous communications (Lawas et al., 2016; Crist & Keijser, 2015; Rawat et al., 2001; Zorn et al., 1999). The intended use for VPN implementations was to allow an organisation's workers to securely access internal network resources from outside of the internal network i.e. remote access. This is achieved through setting up a connection called a tunnel between the user's PC and the organisations servers. VPNs however can also be used as an anonymous communication system in an equivalent manner to an anonymous proxy server. The main difference between the two methods is in the VPN's tunnelled connection. The tunnelled connection between the user and the VPN server is encrypted.

1.1 Detecting and blocking anonymous communications

IP blocking is a basic technique used to combat malicious threats to networks and it is one of the most common techniques for protecting networks (Thomas et al., 2011). Using this method, an IP address or a range of IP addresses can be blocked from accessing resources located on a web server or on an organisation's internal network. The IP block can be rendered ineffective by using proxies or VPNs. The user's IP address is typically sent out as a source IP address in the network packet containing the request to a web server. However, when using a proxy or VPN, this request is first sent to the proxy server which then forwards it on towards the web server. So, the blocked IP address of the user is not actually making any direct contact with the web server running the IP filter. The offending proxy or VPN IP address can be blocked, but this act of blocking the IP address can be made redundant. Upon discovering that their preferred proxy IP has been blocked, the user can simply switch to a different proxy or VPN provider. Unless preventative action is taken, which will cost a significant amount of time and effort, the user can continue to switch in order to maintain their access.

Another method of securing networks is the use of Access Control Lists (ACL). These are usually implemented alongside IP blocking techniques. An ACL scans network traffic and filters which network packets are forwarded on through the network and which are blocked at the router. Each packet is examined and compared to the policies outlined in the ACL to determine whether it should be allowed or blocked (Cisco, 2006). This is a very rigid form of network security that relies on a lengthy setup. Specifying what is acceptable and what is not takes a large amount of time due to the complexity and sheer number of network protocols that exist. Filtering based on the protocols included in the network packets can be rendered ineffective by VPNs due to how they encapsulate protocols within other protocols. Depending on the exact implementation of ACL, the network topology for the entire enterprise network will not be defined so the ACL cannot determine what is a member of the network. Proxies can easily take advantage of this and the ACL is also susceptible to the user switching proxy provider to circumvent any blocks.

Software based packet inspection is another method that can be used to detect and block usage of a proxy or a VPN. Deep Packet Inspection (DPI) is a popular method for securing networks against network packets containing malicious items such as viruses and other malware that are contained within payloads (Dharmapurikar et al., 2003). DPI examines and manages network traffic as it enters the network in a form of packet filtering that identifies, classifies and blocks packets that contain data (such as the aforementioned viruses) within their payload that goes against pre-arranged policies. This examination occurs at checkpoints located around the network and decisions based on rules assigned by an organisation occur in real-time based on the contents of the packet's payload. Previously, packet scanning software had the limitation of only scanning the packet's header, which contains the information necessary for transmission, but does not contain anything related to its contents. By scanning the packets contents, messages and other information can be extracted and used to identify the specific application or service it comes from. The rules that DPI algorithms operate by were string based, however using regular expression matching improves content scanning speeds (Yu et al., 2006). As powerful as DPI can be, it is defeated by packets that make use of encryption to conceal their contents. VPNs are

particularly effective at bypassing DPI as well as some proxies which support HTTPS, bringing the effectiveness of DPI into question (Sherry et al., 2015).

A field of research that has gained traction of late is the use of machine learning algorithms for classification of network traffic (Bujlow et al., 2012; Dainotti et al., 2012; Finamore et al., 2010; Nguyen & Armitage, 2006, 2008). Over the past decade the research and networking communities have investigated and developed several classification approaches based on multiple algorithms. This has come about because the traditional approach of using TCP and UDP network ports to classify Internet applications has become less accurate. Newer applications that are being developed do not have ports registered to them by IANA and instead make use of ports that are already registered to other applications. The exhaustion of IP version 4 addresses has also contributed to this as organisations and application developers move to mitigate the effect (Dainotti et al., 2012).

Classification algorithms typically require training based on previously labelled data. For classification of network traffic, the network packets form the basis of the dataset. The contents can consist of unedited packet headers, with the information contained being used as the training features. They can also consist of statistical information calculated from streams of packets called flows. Efforts to classify Internet applications have largely been successful, with several datasets being created to represent most of the applications available. However, datasets representing anonymous communication systems are mostly non-existent and research into classification of anonymous traffic is still an emerging research area. A major limitation into classification of this type of network traffic is the use of encryption, which renders the payload of packets unusable as a training feature. Using machine learning capabilities and different feature formats, it should be possible to overcome this limitation. Packet header information such as the sequence and acknowledgement numbers and the general size of the data can potentially be used to train a machine learning algorithm. There is also the option of using flow-based features to enhance the potential training and detection accuracy of an algorithm (García-Teodoro et al., 2009).

1.2 Problem Statement

A potential threat to large scale, enterprise networks are connections both from inside and from outside the network which make use of anonymity techniques to hide their identity. Modern network security systems require the capability to identify these connections and to help the network controllers to mitigate the threats that could be posed. If an enterprise network suffers a breach that results in information being stolen, identifying who is responsible can be difficult if anonymity technologies are being used.

With increasing volumes of internet traffic being generated on modern networks, introducing techniques which can help network controllers and administrators identify threats is essential (Cisco, 2017). These techniques should be able to accurately identify threats in real time with a few false positive and negative results as possible.

1.3 Research Goals

This thesis aims to address the limitations of single-hop anonymous communication method classification by proposing a machine learning based approach utilising TCP header information and flow-based TCP statistics. The particular methods investigated will be anonymous proxy servers and VPNs. A key goal in implementing this approach will be high accuracy and keeping the number of false positives and false negatives to an absolute minimum. Classifying legitimate network packets as having originated at an anonymous communication system could be catastrophic to an organisation that depends on high volumes of traffic reaching their site. Similarly, classifying anonymous communication traffic as legitimate could open up an organisations internal network to malicious activity where the identity of the perpetrator is unknown. Having the ability to accurately determine which class the network traffic falls into can be a step towards allowing a network manager to secure an internal network, especially when combined with other security tools.

The overall aims of the research presented in this thesis is as follows:

- Review proxy and VPN architectures and functionality.
- Review current methodologies for detecting proxy and VPN activity (without utilising machine learning approaches).
- Review traffic classification literature to develop knowledge of the area and get an understanding of how machine learning is applied to this problem.
- Review literature based on applying machine learning methodologies to proxy and VPN detection. The goal here will be to gain an understanding of the potential techniques that can be utilised.
- Gather network traffic from multiple proxy and VPN sources to construct two datasets, one for proxy traffic and one for VPN traffic. Both datasets will contain control traffic to facilitate comparison
- Develop a machine learning approach for distinguishing between proxy and non-proxy traffic accurately.
- Develop an approach for distinguishing between VPN and non-VPN traffic accurately.
- Develop an approach for further investigating VPN traffic with a focus on Stunnel.

1.4 Thesis Contributions

The research presented in this thesis provides a substantial and novel contribution to the area of proxy and VPN based network traffic detection and classification. The work has been peer reviewed in three published conference papers (Miller et al., 2015a, 2016, 2018) and has contributed towards journal publications (Miller et al., 2015b and Miller et al., 2019). The primary contributions of the thesis are:

1. The creation of three datasets in the ARFF format containing captured network traffic, one consisting of proxy network traffic and non-proxied control traffic, a second consisting of VPN network traffic and non-VPN control traffic and a third consisting of VPN network traffic tunnelled through Stunnel and non-VPN control traffic.

2. A machine learning based approach for accurately detecting and classifying proxy network traffic.

3. A machine learning based approach for accurately detecting and classifying VPN network traffic.

4. A machine learning based approach for accurately detecting and classifying VPN network traffic that is tunnelled through Stunnel.

1.5 Thesis Outline

Chapters two through to five present the research undertaken and the experimental work involved with chapter six drawing conclusions and suggesting future work. A brief summary of the chapters is outlined as follows:

- **Chapter 2** reviews the current literature with regards to detection of proxy and VPN network traffic. Current proxy and VPN architectures are described along with current network intrusion detection techniques. A review of machine learning is presented with a focus on its potential for network traffic

classification. Current network traffic classification techniques are reviewed and evaluated.

- **Chapter 3** provides the methodology involved in creating a dataset and populating it with network traffic collected using a number of proxy websites. Also included in the dataset is network traffic that was not collected using proxy websites to act as a control comparison. The chapter then progresses to describe the methodology for using the Azure Machine Learning studio to create, train and test an artificial neural network for the purpose of classifying whether the network traffic contained in the dataset originated from a proxy website or not. The results of this testing are detailed at the end of the chapter and they show that the neural network was able to classify a large number of instances correctly.

- **Chapter 4** provides the main methodology used to create a dataset for training and testing and the methodology for creating, training and testing an artificial neural network designed to classify network traffic as originating from a VPN or not. The dataset is populated with network traffic that was collected using an OpenVPN connection and network traffic that was collected when not using a VPN connection. The chapter then describes the methodology for using the WEKA suite of machine learning tools for the creation, training and testing of an artificial neural network. The results are then outlined at the end and they show that the neural network that was created was able to classify most of the testing instances correctly.

- **Chapter 5** concludes the thesis by documenting the main contributions of the research and suggesting potential future research in the area of proxy and VPN based network traffic detection and classification. Future research includes the capture of additional data for the datasets to strengthen the findings and investigation into the use of automatic hyperparameter tuning in Weka to find the best possible setup of the models used. Also included is a discussion on newer VPN targets that could be used instead of OpenVPN such as Wireguard and potential investigation into other types of machine learning algorithm

including Ensemble Learning, Transductive machine learning and deep learning algorithms.

2. Literature review

This chapter provides a background on the technologies involved in the thesis as well as a review of the current literature in detecting the use of both Anonymising Proxies and VPNs. The background information is given in an attempt to give the reader a better understanding of the technologies that are being investigated in this thesis. A review of current literature is given to help show the reader what other research has been done in this area and to show where inspiration is being taken for the techniques being described in Chapter 3.

2.1 Proxies

Web proxy servers are a computer network system or application that acts as an intermediary for requests from clients seeking resources such as files, web pages or other resources from other servers on the internet. They were invented to add structure and encapsulation to distributed systems and to help control complex. Proxies normally operate under 2 different protocols, Hypertext Transfer Protocol (HTTP) and Socket Secure (SOCKS) (Ligh et al., 2010).

HTTP is a protocol that allows a user to send requests for resources on the internet. It is the foundation of data-exchange on the Web. It is a client-server protocol which means that requests are initiated by the receiver, usually a web browser but it could be a robot that automatically explores the internet to populate and maintain a search engine such as Google search. Each individual client request is sent to a server, which handles it and provides an answer, called the response. Between the client and the server there can be other entities. These entities are referred to as Proxies and they perform different operations such as acting as gateways, caches or as a method of anonymising the client. Whilst HTTP is not designed solely for proxy communication, proxies still use it because it supports both encrypted and unencrypted traffic as well as the ability to allow non-HTTP traffic to pass-through a proxy-server.

SOCKS is a protocol that exchanges network packets between a client and server through a dedicated proxy server. It is known as a circuit level proxy intended for use with applications. The SOCKS protocol consists of 3 major versions - SOCKSv4, SOCKSv4a and SOCKSv5. SOCKSv4 is a protocol that is designed for proxy-based applications. The other two versions are extensions of it that provide extra features

and support for other protocols with SOCKSv5 providing support for the user datagram protocol (UDP), IPv6 and strong authentication (Lee et al., 1996).

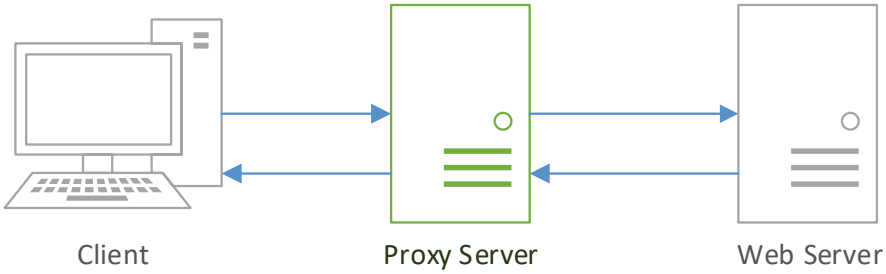

Figure 2.1: HTTP Proxy Interaction

During a normal HTTP interaction, a client will communicate directly with a server over HTTP. When a web proxy is involved the client will instead send its traffic to the proxy, which itself will communicate with the target server on the client's behalf. An example of this is shown in figure 2.1. This means that web proxies have two roles to fulfil themselves, that of a HTTP client and that of a HTTP server. This is because the client is sending request messages to the proxy that are intended for the target web server. The proxy server must be able to handle and process those requests properly and the subsequent responses to facilitate a successful connection with the client. At the same time, the proxy itself must send requests to the target server, therefore it must be able to send requests and receive responses just like a normal HTTP client.

SOCKS proxies operate at a lower level of the OSI layer model than HTTP, as shown in Figure 2.2, and differs in their operation. Where an HTTP proxy acts as a middle man or stepping stone between a client and server by forwarding the HTTP requests, SOCKS proxies relay communications via TCP connections at a firewall gateway to allow a user application transparent access through the firewall (Lee et al., 1996). To use a SOCKS proxy connection, a client must have SOCKS client and server software installed on the user's machine. This can be in the form of an application such as PuTTY[1] or a web browser, or it can be installed in the TCP/IP stack. The client software's main function is to redirect network packets into a SOCKS tunnel. The SOCKS client then initiates a connection to a SOCKS server. The proxy server then

[1] http://www.putty.org/

acts as the client and communicates with an external web server. This external server is only aware of the proxy server and not the original client that initiated the connection.

A SOCKS proxy is different from a HTTP proxy because they are application proxies. For example, when using a HTTP proxy, the HTTP request itself is being forwarded and the proxy server then performs the request on the client's behalf. A SOCKS proxy server doesn't forward request but instead negotiates a proxy connection by exchanging messages between the client and server. When a connection is established, the client communicates with the SOCKS server using the SOCKS protocol. The external server then communicates with the SOCKS server as if it were the actual client.

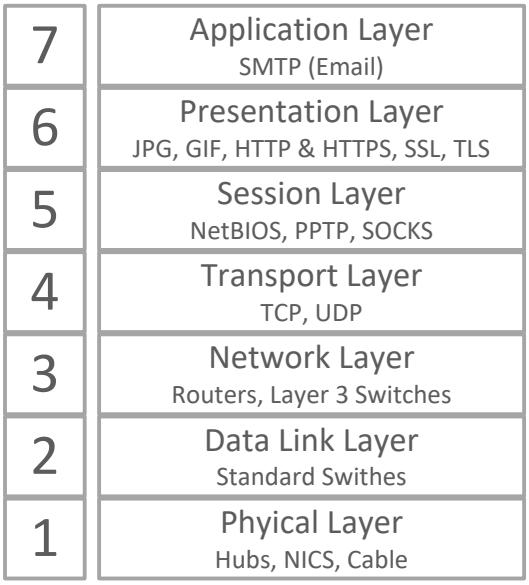

Figure 2.2: OSI 7 Layer Model

There are two overall types of proxy server; those dedicated to a single client and those that are shared among many clients (Gourley & Totty, 2002). Proxy servers that are dedicated to a single client are referred to as private proxies and those that are available to multiple clients are public or shared proxies. Private proxies perform a few specialised tasks, mainly when they are run directly on client computers. ISP services run small proxies to provide certain services such as extended browser features and to host advertising. Public or shared proxies are more common as they are usually

accessible from the Internet. These types of proxy are more popular due to their accessible nature and are easier and cheaper to administer. There are also a number of sub-types of proxy in addition to the two overall types. These sub-type Proxies have a designated role or job. These roles include Content Filters, Document Access Controllers, Security Firewalls, Web Caches, Reverse Proxies, Content Routers, Transcoders and Anonymizing Proxies. A number of these are usually limited to enterprise networks, but some can be found on home networks as built-in modules of the gateway router supplied by an Internet Service Provider (ISP) and others, like anonymizing proxies, can be found openly on the Internet.

2.1.1 Content Filters

Content filter proxies provide administrative control over client transactions as they happen at the application protocol layer of the network stack (Luotonen & Altis, 1994). This is commonly used in both commercial and non-commercial organisations, especially in schools as a method to control access to the Internet. Requests may be filtered using several methods, such as URL blacklists, URL *regex* filtering or content keyword filtering. More in-depth filtering can be accomplished by analysing the content of requests to discern whether they should be allowed or not. If the requested URL passes these filters, the filter proxy then proceeds to fetch the website, usually over HTTP. On the return path, dynamic filtering can be applied to block specific elements of a webpage, such as embedded videos or JavaScript. A drawback to content filtering proxies is that they cannot, normally, scan websites that are transmitted over an encrypted, HTTPS session where the chain of trust for the website in question has not been tampered with. The chain of trust refers to the use of certificates in the implementation of SSL/TLS connections. These certificates are issued by root Certificate Authorities (CAs) who can attest to the legitimacy of the website. However, content proxies can generate their own root certificate and inject that into the communications as a trusted root certificate. With this certificate in place, the content filter can decrypt requests in order to scan the normally encrypted content. In this situation, the filter is effectively operating what is known as a Man in The Middle (MiTM) attack.

2.1.2 Document access controllers

Document access controllers are proxy servers that have the role of implementing a uniform access control policy across a larger network of web servers and resources (Gourley & Totty, 2002). This eliminates the need for multiple access control systems and simplifies the administration of access control as all the controls can be configured on a centralized proxy server. This is particularly useful when used in a large data centre that makes use of servers of many different types and models which all have slightly different methods of applying access control.

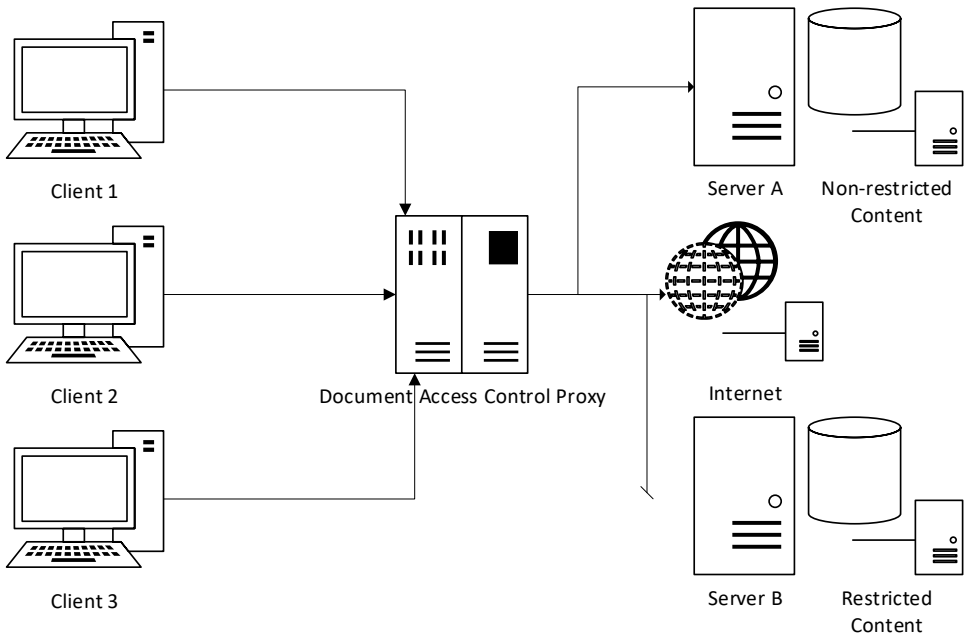

Figure 2.3: Document Access Controller Example

Figure 2.3 shows an example network which contains a Document access controller. It shows three client pcs that are connected to the controller and it show some example content that the proxy controls access to. Server A contains non-restricted content therefore all of the clients can access this resource. Some of the clients will require access to the Internet and the controller regulates this access, only allowing the clients that have the required permissions to access it. Server B contains restricted content and by default none of the clients can access this unless they have been authenticated and have the required permissions.

2.1.3 Security Firewalls

A significant security problem for business type networks is hostile or unwanted access by users or software (Stallings & Lawrie, 2008). Unwanted user access (an intrusion) can be in the form of unauthorised logon to a machine or gaining the ability to perform higher privilege actions than what is normally authorised. Unwanted software access can take the form of a virus, Trojan horse or other form of malware (Wang et al., 2013). A firewall is defined as a component or set of components that restrict access between a protected network and external networks (Kumar et al., 2014). There are a few different types of firewall. These are: packet filtering firewalls, stateful inspection firewalls, application-level gateway, circuit-level gateway and proxy firewalls (see figure 2.4).

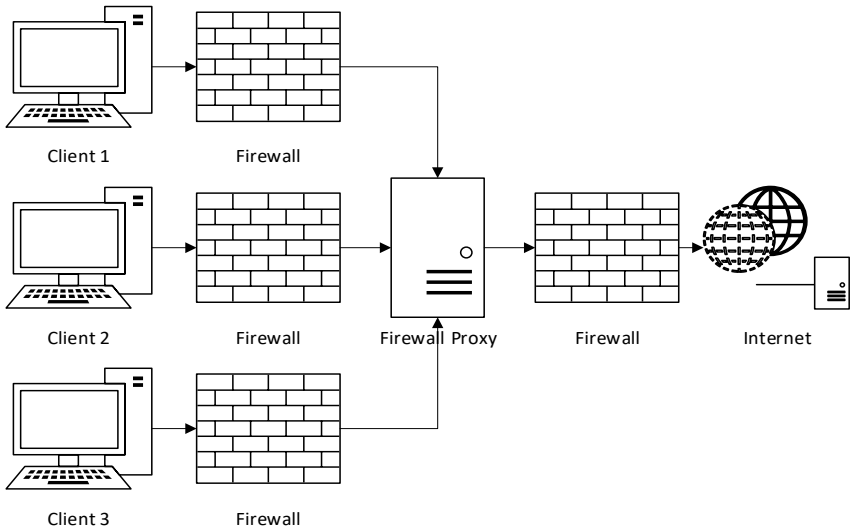

Figure 2.4: Firewall Proxy separating clients from Internet

Packet filtering firewalls apply a set of rules to incoming and out-coming IP packets, any packets that adhere to those rules are forwarded on to their destination and any that don't are discarded (Ali et al., 2015). A stateful inspection firewall reviews the same packet information as a packet filtering firewall, but also records information about the TCP connections that are sending and receiving the packets as well. Some also keep track of the TCP connection sequence numbers to prevent attacks that depend on the imposters using a sequence number, such as session hijacking (Ali et al., 2015).

An Application-level gateway acts as a relay for application-level traffic. A user will contact the gateway using a TCP/IP application such as FTP or Telnet and the gateway asks for the name of the remote host to be accessed. When the user responds with the name of the remote host and provides a valid ID and authentication information, the gateway contacts the application on the remote host and relays application data between the two. If the gateway does not support the application, the service is not supported and cannot be forwarded across the firewall. Application-level gateways tend to be more secure than packet filtering firewalls as they only scrutinise a few allowable applications. A circuit-level gateway can be a stand-alone system or it can be part of a specialised function performed by an application-level gateway. A circuit-level gateway operates in much the same way as an application-level gateway however, once it sets up the TCP connections, it does not examine the contents. The security function consists of determining which connections will be allowed (Ali et al., 2015). Proxy firewalls are a network security application that filters network packets at the application layer and they are the most secure type of firewall at the expense of speed and functionality of the network because they can limit the applications that are supported on the network. Proxy firewalls act just like standard proxy servers in that they act as an intermediary between a client computer and a destination web server. They are also the only machine on a proxy firewall protected network to have a direct connection to the Internet. This means that any other machine that wants to access a resource from the Internet will have to use the proxy firewall as a gateway. As the proxy firewall receives every request travelling to and from the network, it is able to filter and log requests based upon inspection of their packets. An added benefit to this type of proxy is that content that is being requested by multiple clients can be cached locally to increase the access time for the content. However, on networks with large amounts of traffic, the proxy firewall could be the cause of a reduction in performance due to the creation of a bottleneck or increasing the risk to the network by becoming a single point of failure.

2.1.4 Web Caches

Alongside security proxy firewalls, there are also dedicated web caching proxy servers. Web caching is the temporary storage of popular remote web resources on a

local server (Ponnusamy & Karthikeyan, 2013; Singh et al., 2011). These proxy caches are used to reduce the strain of repeated requests for the same resource on web servers and network bandwidth providers (Cobb & ElAarag, 2008). Figure 2.5 shows a common Web Cache Proxy setup. Caching proxies can be configured in one of two ways. The first way positions one or more servers on the network between a web server or (more commonly in a datacentre) a group of web servers and incoming network traffic from the Internet. This is designed to reduce the load on the web servers.

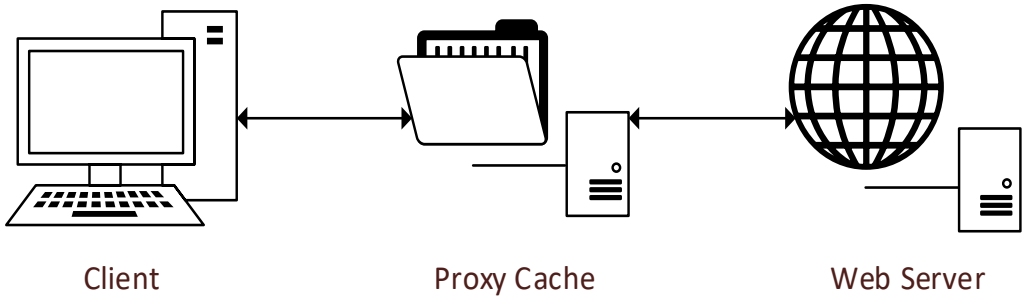

Figure 2.5: Standard setup of a Web Cache Proxy

The second configuration is designed with a focus on reducing congestion on the network. With this approach the proxy cache is located on the same network as the client machines making requests. As the request comes in, the proxy first determines if it has the requested material is stored locally. If the material is stored locally, it replies to the quest and delivers the material. If not, it initiates a connection with the web server to access the material and fetches the materials on behalf of the client and potentially caches it.

2.1.5 Reverse Proxy

A reverse proxy is a server that transparently hands off requests to another server (Reese, 2008). Contrary to the normal operation of a proxy server where the server acts as an intermediary between clients and servers, the reverse proxy itself appears to the client as a web server, acting as an intermediary for its associated servers to be contacted by any client even if the servers are behind a firewall. An example can be seen in figure 2.6.

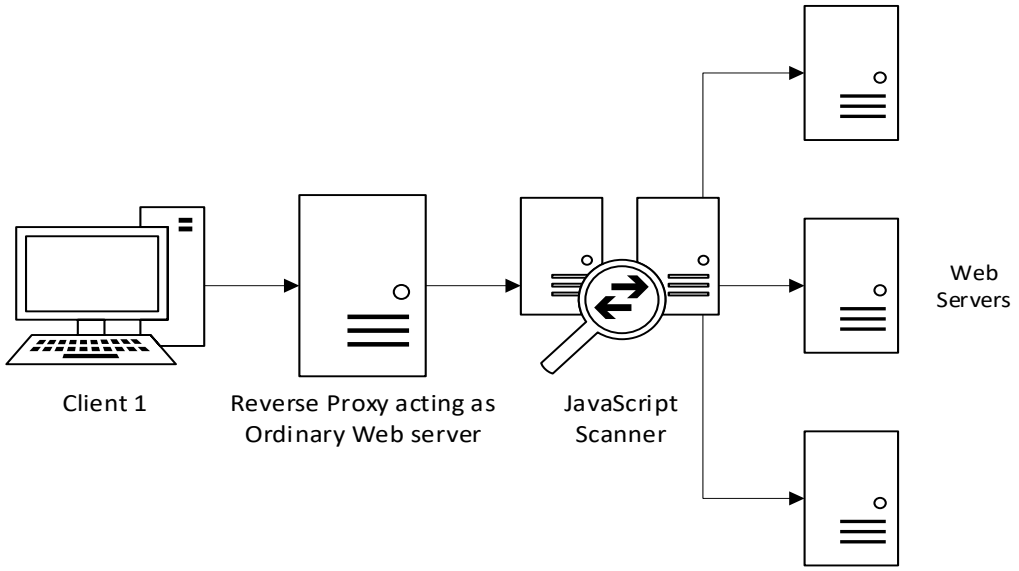

Figure 2.6: Reverse Proxy with JavaScript Scanner acting as a load balancer

Reverse proxies can also be used as a load-balancer among several back-end servers or to provide caching for a single server to reduce the load of commonly requested data on the target server. In this latter implementation the reverse proxy can be referred to as a server accelerator (Gourley & Totty, 2002). The popular Content Distribution Network (CDN) and Distributed Denial of Service (DDoS) protection company Cloudflare provides its services by acting as a reverse proxy (Durumeric et al., 2017). When a client visits a site that is protected by Cloudflare's services, instead of connecting directly to the web server that is hosting the website, the client connects to one of Cloudflare's servers which serves a cached version of the site or proxies the connection to the origin server. Another use case of reverse proxies can be in the mitigation of attacks against websites. (Wurzinger et al., 2009) describe their method of mitigating Cross Site Scripting (XSS) attacks using a reverse proxy to relay traffic to and from the web server that is being protected. Each response made by the web server is forwarded by the reverse proxy to a JavaScript scanning component which scans the response for harmful scripts. If detected the proxy blocks the response from being delivered and instead notifies the client of the attempted attack.

2.1.6 Content Router

Content Routers are proxy servers that have the ability to redirect requests as part of an information-centric network (ICN). In a simple ICN setting, requests for content

are generated and sent by end users. Each request consists of a content name, whole controls access to the content and the location from which it can be accessed (Kurose, 2014). These requests are forwarded among content routers towards the location and controller of the content. Before a content router forwards a content request, it first checks its own local cache store for the requested content. If the content router has the content stored locally, the router itself can satisfy the request for that content, sending it to the requestor. As content is sent along its path from controller to requestor, each content router on the way stores a copy of the content in its local cache. This ensures that repeat requests for that same content can be detoured to content routers which may have the requested content, helping reduce the congestion on the network that would be created by having to forward requests directly to the content's controlling server (Wong et al., 2011).

2.1.7 Transcoder

Transcoding can be defined as the transformation that is used to convert a multimedia object from one form to another (Chang & Chen, 2003). Based on where the actual transcoding takes place, different technologies can be classified as belonging to server-based, client-based or proxy-based approaches (Cardellini et al., 2000). Transcoding proxy servers can modify the format of content before it is delivered to the destination. They can convert images from one filetype to another to reduce size and modify the image itself to make it fit onto different types of screen, for example when accessing a desktop website from a smartphone or similar device. They also have the ability to modify text files in a similar method, even translating the text into different languages based on the country ID of the user requesting the content (Gourley & Totty, 2002). This can be particularly useful when attempting to provide content for an international community where everyone may not understand or speak the original language of the content.

2.1.8 Anonymous Proxies

Anonymity technologies allow Internet users to maintain their privacy by preventing the collection of identifying information such as IP addresses. Due to an increasing awareness of what is shared and collected online, Internet users are growing more concerned with their privacy and are turning to the use of technologies such as

anonymous proxies (Li et al., 2013). Anonymous Proxies are one of the easier to deploy technologies for anonymity on the Internet and are generally access through a web browser (Edman & Yener, 2009). They are proxy servers that are based on the open Internet and are accessible to the general userbase of the Internet. The aim of an anonymous proxy is to make a user's Internet activity untraceable by acting as an intermediary between the user's client pc and the rest of the Internet. Anonymity is provided by the proxy server by processing client requests using the proxy server's IP address rather than the user's IP address. The server relays requests from the user to their destinations and delivers the responses back to the user. This provides a basic level of anonymity. However, the proxy servers can see both the source (client IP address) and destination (resource IP address) and therefore can track user activities and what is being relayed between the source and destination.

There are multiple ways of setting up a proxy server. Two examples of some of the more popular technologies for setting up proxy servers are PHP and Common Gateway Interface (CGI) based scripts. Both provide the required functionality that anonymous proxy servers rely on and they have the benefit of being supported across multiple operating systems. Glype is a PHP based script and is one of the most common and popular web proxy scripts available on the Internet. Setting up a proxy server using the Glype proxy is accomplished by downloading the script files from the Glype website and then relocating those files to the correct directories on the webserver. This may appeal to user's who have access to web capable servers that are located on a different local network, such as owning a Virtual Server hosted by a server hosting company. However, a simpler option would be to access one of the many existing proxy sites already available. A study done in 2011 on the geo-location of public proxy servers found that there were 7,246 proxy servers available (Li et al., 2013). A list found on the Glype proxy website listed 3,389 unique servers that were running the Glype script (Miller et al., 2016). A more recent list showed that there were 50,824 individual web proxies[2]. This presents a problem when trying to block access to these proxies because there are so many that when one server is blocked, it's a simple case of accessing another proxy server. The difficulty lies in compiling a complete list to add to an IP block list or Access Control List. Due to the ease of setting

[2] https://proxy.org/web_proxies.shtml

up new proxy servers, new proxy servers are being added all the time. URL filtering is defeated by the proxy server's use of encoding to obfuscate and hide the actual URL from the filters.

For example, when encoding is applied, the URL:
http://www.radiocarb.com/p/browse.php5?u=https://www.wikipedia.org/
becomes:
http://www.radiocarb.com/p/browse.php5?u=czovL3d3dy53aWtpcGVkaWEub3JnLw%3D%3D&b=13

This is an example of the base64 encoding scheme, however some PHP based proxies also make use of the simple ROT13 encoding scheme, which is based on the Caesar encryption cipher using a key of 13. CGI proxies make use of the Common Gateway Interface which is a standardised protocol created to enable web servers to execute console/terminal style applications. The most common use of these is to generate web pages dynamically each time a request for that web page is received. CGI proxies use this type of script to perform the act of proxying a connection. Proxy clients send a request containing the URL of the website they wish to visit embedded in the data portion of an HTTP request. The proxy server pulls the destination information from the embedded data and uses it to send its own HTTP request to the destination. Whenever the result is returned from the destination web server, it is forwarded to the proxy user (Leberknight et al., 2010). An example of a CGI proxy script that is available for download is CGIProxy[3] by James Marshall (Marshall, 2002). While Glype proxies enable URL obfuscation by default, the CGIProxy script does not. ROT13 encoding can be enabled by removing the line comments for the methods proxy_encode() and proxy_decode() in the script. The script also provides support for custom encoding code to be added such as hexadecimal encoding.

2.1.9 Conclusion

This section of the literature review has provided background on the different types and functions of Proxy servers, highlighting that there are a number of legitimate tasks

[3] https://www.jmarshall.com/tools/cgiproxy/

that proxy servers accomplish some of which are necessary to the continued operation of the internet. Details are given on how each different proxy task operates, what the results of the task are and how the results affect the internet with some examples given of real-world application. Anonymising proxies are detailed at the end of the section providing a background into how some of the more popular anonymising proxy scripts operate, highlighting how easy it is to access or even set up an anonymising proxy server. This type of proxy is the focus of this thesis as the other types of proxy are seldom used to commit criminal acts. Also described is the methods that the scripts use to hide themselves from detection through the use of URL obfuscation via encoding algorithms. Techniques that typically block access to websites fail to keep up with how quickly new proxy servers can be created which lends weight to the argument that new techniques need to be developed to detect the use of proxies and enable administrators to take effective action.

2.2 Virtual Private Networks (VPNs)

2.2.1 Introduction

A Virtual Private Network (VPN) is the use of varied techniques to provide private networks of resources and information over any public network (Hawkes-Robinson, 2002). They enable organisations and individuals alike to connect their resources over the Internet, but control access to those resources by only making them available to those that are part of the VPN. Normally, without the use of a VPN, to achieve such a private connection would require a great investment in time, finances and work to setup a dedicated line of communication. Instead, by using VPNs it is possible to extend private networks and allow the sharing of data as if the computing devices attached to the VPN were all directly connected to a local network (Mason, 2004).

The data of the private network is said to be “tunnelled” inside a public network packet (Hawkes-Robinson, 2002). It enables a remote machine on network X to tunnel traffic, that might not normally be able to be sent across the Internet, to a gateway machine on network Y and appear to be sitting, with an internal IP address, on network Y. The gateway machine receives traffic to this internal IP address, and sends it back to the remote machine on network X (Schneier & Mudge, 1998). This itself does not provide

much security. Intercepting these tunnelled packets would still allow for the contents of the private packets to be intercepted and exposed by a third party.

To overcome this, the private packets need to be encrypted and above that, some form of authentication needs to be used. VPN protocols vary in their support for encryption and authentication schemes. Each of the following sections will discuss some example algorithms and schemes supported by each VPN protocol.

2.2.2 PPTP

The Point-to-Point Tunnelling Protocol (PPTP) is a link layer VPN protocol that is designed to tunnel Point-to-Point Protocol (PPP) connections through an IP network, creating a VPN connection (Zorn et al., 1999; Schneier & Mudge, 1998). PPTP encapsulates the virtual network packets inside of PPP packets, which are then encapsulated in Generic Routing Encapsulation (GRE) packets (Farinacci et al., 1994). The final packets are sent over IP from the client to the gateway PPTP server and back again. PPTP does not provide any methods for keeping data confidential or for providing strong authentication. The Microsoft implementation that was included with Windows NT provides a framework for negotiating authentication and encryption algorithms between server and client which relies upon existing negotiations contained within extensions and enhancements of PPP (Simpson, 1996). Some example authentication algorithms are the Password Authentication Protocol (PAP), the Challenge-Handshake Authentication Protocol (CHAP), MS-CHAPv1/v2, Microsoft's implementations of CHAP, and Extensible Authentication Protocol (EAP). CHAP and MS-CHAPv1/v2 have faced extensive scrutiny over the years (Microsoft, 2012; Schmidt, 2012; Hawkes-Robinson, 2002; Schneier et al., 1999; Schneier & Mudge, 1998). PAP transmits the username and password from the client through an unencrypted channel which leaves it vulnerable to eavesdropping attacks. This leaves it in the position where it can only be used as a last resort. Due to the vulnerabilities that have been found in the authentication and encryption algorithms it uses, PPTP does not see widespread use anymore.

2.2.3 L2TP

The Layer 2 Tunnelling Protocol (L2TP) is also a link layer VPN that extends the PPP model by combining features of PPTP with features of the Layer 2 Forwarding (L2F) protocol. (Townsley et al., 1999). L2TP functions similarly to PPTP. Higher level protocols, commonly PPP connections, are encapsulated within an L2TP tunnel by setting up an L2TP session. The L2TP packets in turn, including both the payload and the L2TP header are transported within a UDP packet. L2TP is also similar to PPTP in that it does not provide any methods for confidentiality or authentication and instead inherits existing protections from PPP. A protocol suite called IPsec was introduced to provide improved authentication and confidentiality over the PPP methods (Patel et al., 2001). The original PPP methods used by L2TP were found to be vulnerable to a Denial of Service (Dos) attack which involved transmitting a request to stop the connection using the correct identification in order to terminate the VPN session (Kara et al., 2004). This was a vulnerability that was solved in an updated version of L2TP called L2TP version 3 (L2TPv3). The new version included an optional authentication and integrity check that nullified the vulnerability. L2TP is often combined with another authentication and encryption protocol suite called Internet Protocol security (IPSec) (Kent & Atkinson, 2005).

2.2.4 IPsec

IPsec includes a collection of standardised protocols for mutual authentication between two hosts at the beginning of a VPN session and for the negotiation of cryptographic keys used to enable encryption for the session (Kent & Atkinson, 2005). Data is kept secure by authenticating network packets to make sure of the integrity of the packet and that encapsulation has been implemented correctly. There are two modes in which IPsec can provide this functionality: transport mode and tunnel mode (Berger, 2006). In transport mode, the original packet is edited to include a new IPsec header in the original IP header. This additional header contains the information needed to perform authentication and integrity checking. In comparison, tunnel mode provides more flexibility. In tunnel mode, the entirety of each original IP packet is encapsulated inside a new IP packet consisting of a new IP header and the IPsec header (Kent & Atkinson, 2005). This adds a layer of abstraction from the original IP packet's

contents therefore providing confidentiality for the payload. To determine which mode is to be used during a connection, security information defining the modes that each end point supports needs to be exchanged. This is referred to as a security association (Berger, 2006). It contains information on the mode of IPsec to be used, the encryption algorithms to be used and the encryption keys used to set up the encryption. Exchange of this information is completed using the Internet Key Exchange (IKE) protocol (Harkins & Carrel, 1998).

2.2.5 IKE

IKE is used as part of IPsec to negotiate and establish connections by sharing authentication data and encryption keys between two hosts (Harkins & Carrel, 1998). In IKE version 1, IKE messages are sent between the hosts using UDP packets on port 500 and form the basis of a two-stage negotiation. This exchange of messages relies on the *Internet Security Association and Key Management Protocol* (ISAKMP) (Maughan et al., 1998). The first stage involves the setup of the IPsec security association. At this point there is no encryption of data or authentication of either host. Therefore, the two hosts attempt to authenticate themselves by sending their respective encryption public keys via the Diffie-Hellman (DH) key exchange method (Diffie & Hellman, 1976). Once the keys have been exchanged and the two hosts have been successfully authenticated, stage one is complete and stage two begins. In stage two of the negotiation, the two hosts work out the parameters for the VPN tunnel or tunnels that will be setup. These include the symmetric encryption keys and their expiry information, the security policies of the connection, the network routes and other information pertaining to the connection. Once worked out, the connection between the two hosts will be complete and data can be exchanged in a secure way (Berger, 2006). Internet Key Exchange version 2 (IKEv2) is an update that combines the contents of the multiple protocols and methods the IKEv1 uses to accomplish its tasks into one overall standard (Kaufman et al., 2014).

2.2.6 Secure Socket Layer (SSL)-based VPNs

Secure Socket Layer (SSL)-based VPNs operate on the transport level of the OSI network layer model as opposed to IPSec, L2TP and PPTP which operate on the link layer. This is due to their use of SSL/TLS to provide authentication and confidentiality and HTTPS for transferring data. Reliable transmission is available without any extra effort due to the position of SSL/TLS on the network layer model as TCP is also located on the transport level (Rowan, 2007). SSL VPNs are often called clientless VPNs because they do not need any additional client software to be installed in order to use them. They however do rely on web browsers to handle the client side of the tunnel as most web browsers have SSL protocol support built in. This has an added benefit of making the SSTP VPN platform agnostic, enabling users to access resources from a variety of platforms running on different operating systems. Use of HTTPS enables connections to be made through most firewalls.

One example of an SSL-based VPN is Microsoft's Secure Socket Tunnel Protocol (SSTP) (Jain et al., 2011). SSTP provides an encrypted tunnel by means of the SSL/TLS protocol. PPP network traffic is encapsulated in this tunnel and transferred over a HTTPS. When a client establishes an SSTP-based VPN connection, it first establishes a TCP connection to the SSTP server over TCP port 443. The SSL/TLS handshake process used for transferring keys and authenticating and encrypting the connection occurs over this TCP connection. After the successful negotiation of SSL/TLS, the client sends an HTTP request with content length encoding and a large content length on the SSL protected connection. The server sends back an HTTPs response with the *HTTP 200 OK* status if everything is in order. Once the HTTPS connection is established successfully the client can send and receive SSTP Control packets and SSTP Data Packets. SSTP control packets contain messages to negotiate parameters and to ensure there is no untrusted man-in-the-middle (MITM). SSTP data packets contain the encapsulated PPP traffic as a payload.

2.2.7 OpenVPN

OpenVPN[4] is a well-known and popular VPN protocol (Feilner, 2006). Due to its very simple configuration and the mixture of enterprise-level security, usability and other features, plus its support for most of the operating systems that are available, it is widely regarded as among the best VPN solutions (Pohl & Schotten, 2017; Crist & Keijser, 2015). It falls loosely into the SSL-based VPN category due to its use of the SSL/TLS protocol to secure connections. However, OpenVPN also makes use of Hash-based message authentication codes (HMAC) in combination with the SHA1 hashing algorithm for ensuring packet integrity. OpenVPN has two authentication modes. In mode one a pre-shared static key is used to provide authentication and encryption. In mode two, SSL/TLS mechanisms are used for authentication and key exchange[5] (Feilner, 2006). In static key mode, a pre-shared key is shared between both hosts before the tunnel is set up. This static key contains four independent sub-keys: HMAC send, HMAC receive, encrypt and decrypt. The preferred mode of operation is mode two which uses SSL/TLS. In this mode an SSL session is established requiring both hosts to present their own authentication certificate. If the authentication of the hosts succeeds, negotiation and exchange of the encryption/decryption and HMAC keys begins. Rather than the keys being static as in mode 1, in mode 2 the keys are randomly generated either by OpenSSL's *RAND_bytes* function or by using the TLS pseudorandom function (PRF) alongside random source material from both hosts. The keys are then exchanged over the SSL/TLS connection and the tunnel forwarding process begins. The data to be encrypted and transferred in the tunnel includes a 64-bit sequence number and the payload data consisting of an IP packet or Ethernet frame. Encryption of the tunnel packets is carried out using the Blowfish secret key block cipher (Schneier, 1994). OpenVPN then multiplexes the SSL/TLS session that is used for authentication and key exchange with the encrypted tunnel data. SSL/TLS is designed to operate using a reliable transport protocol so OpenVPN provides a reliable transport layer on top of UDP. The actual IP packets are tunnelled over UDP without an added reliability layer after they have been encrypted and signed with an HMAC as the IP packet forwarder has been designed to operate over an unreliable transport layer.

[4] https://openvpn.net/
[5] https://openvpn.net/index.php/open-source/documentation/security-overview.html

2.2.8 Conclusion

In this section a detailed background is given on various VPN technologies which shows how VPNs have been developed and improved over time. The information provided helps explain why VPNs were developed is and how they operate. It also shows that there are multiple uses for VPNs when it comes to computer networks. Some of these uses are entirely legitimate and necessary to the continued functioning of business networks by allowing employees to access company resources over a secure connection from a remote location. There are however criminals who make use of these same technologies to hide their identity to avoid getting caught when committing cyber-crimes. For this reason, there is a need to develop techniques to detect the usage of VPN technologies to add to the other network information gathering and recording tools that are available. This thesis will focus on OpenVPN as it is easy to set up a VPN server and to then make use of its capabilities. This will form the basis for further research into other more complex techniques.

2.3 Intrusion Detection

Intrusion detection is the process of monitoring connections coming to and leaving from a computer or network and then analysing those connections for signs of potential violations or incidents that go against security guidelines and acceptable use policies (Scarfone & Mell, 2007). Causes of these incidents can include attackers gaining unauthorised access to systems, malware such as spyware and Trojan viruses and misuse of system privileges by users or attempts to gain additional privileges. An intrusion detection system is the software that automates this process. When detecting possible incidents, an IDS can take several actions. One would be to report the incident to a system security administrator, who could then initiate a response to mitigate the effects of the incident. Alongside alerting an administrator, the IDS could also keep a record of incidents that could be referenced later and to help prevent future cases of that incident.

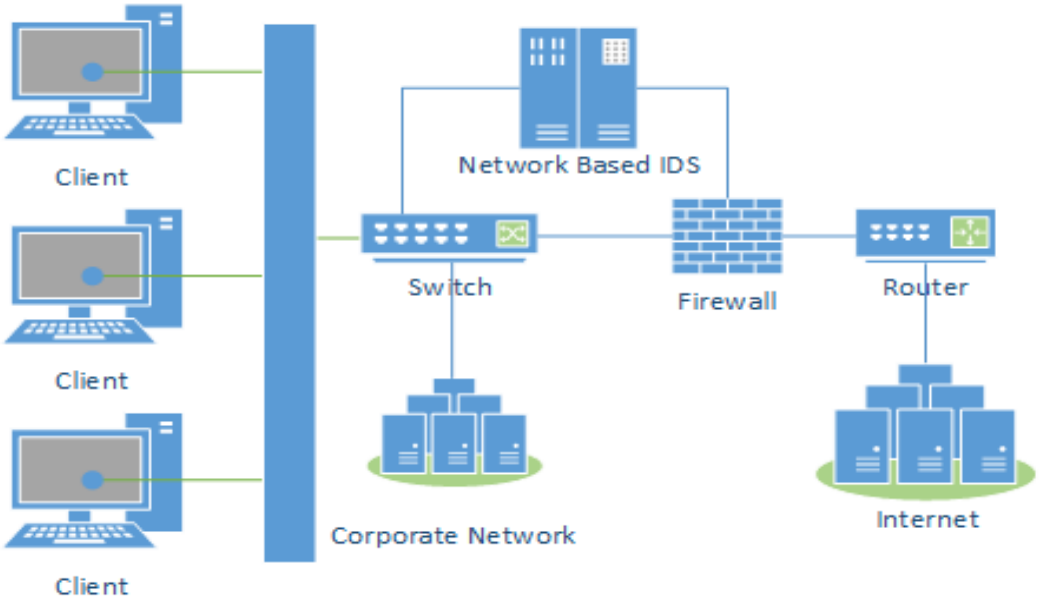

Figure 2.7. Network Based Intrusion Detection System

There are several different types of Intrusion Detection System (IDS) which can be classified as Network based, Host based, Network Behaviour and Wireless (Scarfone & Mell, 2007). Network based systems monitor the traffic of a network using sensors placed at certain parts of the network and IDS management servers. They analyse the activity recorded by the sensors to identify incidents of intrusion. Figure 2.7 shows the typical layout of a network that includes a network-based IDS. Host based systems differ from network-based systems by monitoring a single host. Network Based Analysis (NBA) systems monitor network traffic in order to identity threats that generate unusual traffic flows such as malware or port scanning attempts. Wireless IDSs apply similar techniques to network-based systems specifically to wireless network traffic that makes use of wireless networking protocols.

2.3.1 Machine Learning in Intrusion Detection Systems

Integrating machine learning techniques into IDSs is a method of increasing the power and accuracy of the detection system. Machine learning techniques include various kinds of artificial neural networks and classification techniques as well as genetic algorithms and fuzzy logic. There has been various research studies looking into integrating machine learning into IDSs with the recent trend being improving the machine learning aspect by combining different techniques to increase detection

accuracy and to decrease the computational effort required to train the systems. (Lin et al., 2015) proposed a feature representation technique using a combination of the cluster centre and nearest neighbour approaches. Experiments that were carried out made use of the KDD-Cup99 dataset and showed that the approach required less computational effort to provide similar levels of accuracy to k-NN. (Xiang et al., 2008) proposed a multiple level hybrid classifier that combined supervised tree classifiers with unsupervised Bayesian clustering. Performance of this approach was also measured using the KDD-Cup99 dataset and experiments showed that it provided a low false negative rate of 3.23% and a false positive rate of 3.2% with a high detection rate for both known and unknown attacks. (Khan et al, 2007) made use of a Support Vector Machine (SVM) for classification and a clustering tree technique called Dynamically Growing Self-Organising Tree (DGSOT) to improve the training times of the SVM. Experiments were carried out using the DARPA98 dataset and showed that using a clustering tree helped to increase the accuracy rate of the SVM and lower the rates of false positives and false negatives.

(Özyer et al., 2007) provided a system that made use of both genetic algorithms and fuzzy logic to create a genetic fuzzy classifier to predict different behaviours in networked computers. Their results showed that there was a benefit to using fuzzy logic to pre-screen rules before classifying with the genetic algorithm as it decreased the time needed to train the system. However, the systems accuracy in detection did not show much increase and showed a decrease in accuracy in some classes compared to other approaches. An earlier study used 3 different anomaly detection techniques for classifying program behaviour (Ghosh et al., 1999). These techniques were an equality matching algorithm for determining what was and wasn't anomalous behaviour, a feed forward backpropagation neural network for learning the program behaviour and the third being a recurrent neural network called an Elman network for recognising recurrent features of program behaviour. Their study showed that the performance of intrusion detection benefited greatly from the use of the backpropagation network and the Elman network. The consensus that can be gathered from these studies is that the use of machine learning techniques does improve the accuracy and performance of intrusion detection systems.

2.4 Machine Learning and Neural Networks

2.4.1 Machine Learning Methods

Machine learning is an area of study that evolved from research into the areas of pattern recognition and computational learning theory with regards to artificial intelligence with the term being coined for the first time in 1959 by Arthur Samuel (Samuel, 1959). Machine learning research aims to explore the construction and development of algorithms that can learn from the massive sources of data that surround us. Such algorithms operate by building a model based on making predictions or decisions determined by the inputs that it receives, rather than following a strictly programmed set of instructions.

There are two broad categories of machine learning which depend on how the training data is constructed and presented to the learning algorithm. These are known as supervised learning and unsupervised learning. Supervised learning involves the learning of a function that maps the values of a given input to an output based on example input-outputs in a labelled dataset (Khriplovich & Pomeranskii, 1998). The algorithms task when being trained with supervised learning is to learn the most efficient way in which to map a set of inputs to a set of outputs based on examples of inputs and their desired outputs, otherwise known as training data (Russel & Norvig, 2010). To test whether the algorithm generalises well based on the training data, a set of data that is distinct to the training set is typically kept back and used as a form of test dataset. In unsupervised learning, the algorithm is not present with any training data and is left on its own to find a structure to the inputs that it is receiving. The most common machine learning task that involves unsupervised learning is that of clustering where data is given to the algorithm as input and it groups instances of the data together in clusters based on their attribute or features (Russel & Norvig, 2010).

Clustering is just one of many different applications that machine learning can be applied to. There is also classification and regression. In classification, inputs are divided into classes, typically 2 (known as binary classification) however there can be more classes depending on the data. The goal of the machine learning algorithm is to produce a model that can assign a class label to new, unseen data based on the patterns

that it has learned from being trained in a supervised way. Regression is another example of a supervised learning approach. The difference between it and classification is that the output from a regression algorithm can be continually updated with different values rather than being limited to a set of class labels. The purpose of regression is to learn patterns from input data and then use the patterns it has learned to produce predicted values for future instances of the data.

2.4.2 Neural Networks

Neural networks are defined as an interconnected system that produces an output pattern when presented with an input pattern (Wade, 2010). In computing, Artificial Neural Networks (ANN) are learning algorithms that are inspired by the biological neural networks that make up the majority of animal brains and they deal mostly with the problem of classification (Haykin, 2004). The first instance of a mathematical model that is considered to be a neural network loosely based on neuroscience is called threshold logic (McCulloch & Pitts, 1943). This model led to the creation of two approaches to neural network research: one focusing on researching the biological processes of the brain and one focusing on the application of neural networks to artificial intelligence.

A learning hypothesis based on neural plasticity became known as Hebbian learning (Attneave, 1950). This is an example of unsupervised learning. In 1958 an algorithm called the Perceptron was created (Rosenblatt, 1958). This was an algorithm devoted to pattern recognition and is an example of a supervised learning algorithm. The perceptron is an example of a single layer neural network. This is a network comprised of an input layer and a single layer of perceptron neurons.

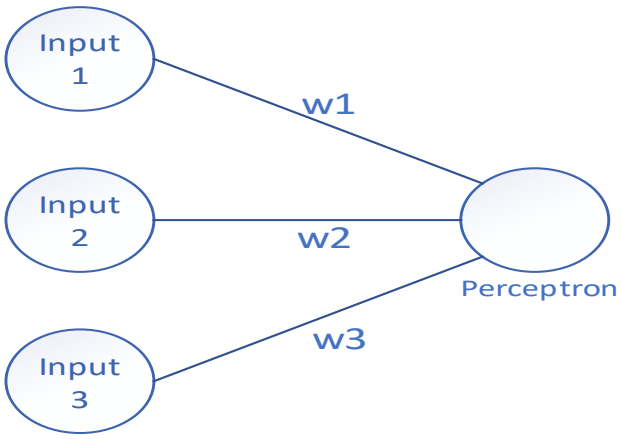

Figure 2.8: Example of a single layer neural network with a single neuron

In figure 2.8, 3 inputs with 3 weights are shown. These feed into the perceptron neuron which modifies the values using the weights provided. It then sums together all the values along with an added offset called bias. Finally, the summed signal is sent to an activation function which determines what the output is.

Figure 2.9 shows a fully connected multilayered perceptron neural network. The biggest difference between the single layered network and the multilayered network is in the addition of the hidden layer in the middle which contains five perceptrons. The output layer contains two perceptrons, signifying that there are two possible classification results i.e. binary classification. Each of the perceptrons in this model function the same as the single layered model by summing the weights and bias, passing the result to an activation function and then passing that on. The difference this time is that the output layer takes the adjusted values from the hidden layer and input.

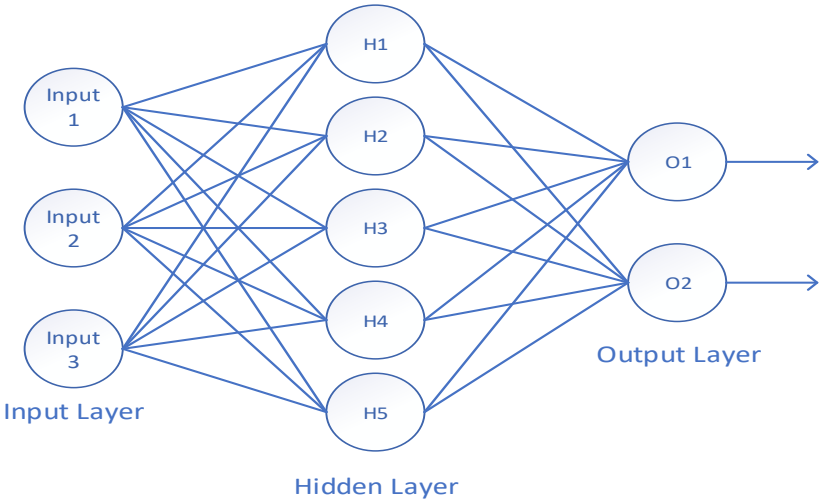

Figure 2.9: Multi-layered Perceptron neural network

Research into neural networks stagnated in 1959 after two key discoveries with the computational machines that processed neural networks (Minsky & Papert, 1972). It was found that the basic perceptrons that were being used were incapable of processing an exclusive-or circuit and that the machines used to process the neural networks struggled when presented with larger networks. In 1975 the exclusive-or problem was solved through the introduction of the backpropagation algorithm (Werbos, 1975). This had the bonus of accelerating the rate at which neural networks were trained. Backpropagation calculates the error difference between the input and output layers and passes the value back through the layers of the network. The error is used to repeatedly adjust the weights of connection in the network to minimise the difference between the input and the output, therefore reducing the size of the error (Rumelhart et al., 1985). In the model shown in figure 2.9, back propagation is used to adjust the weights for the connections between the output and hidden layers will be modified first, then the weights between the hidden layer and input layer.

Recent research has focused on the effect that adding more hidden layers has on the result of a neural network classification. Neural networks that have more than one hidden layer are referred to as Deep Neural Networks (DNN) (Schmidhuber, 2015; Bengio, 2009). The extra layers enable the model to perform classifications on complex non-linear data. Two implementations of a DNN are the recurrent neural network and the convolutional neural network (Krizhevsky et al., 2012; Mikolov et al., 2010).

2.5 Conclusion

Chapter 2 has provided a background on the technologies and techniques used in the rest of this thesis as well as providing a review of the current literature in detecting the usage of Anonymising proxies and VPNs. The technical background provided allows for an understanding of how Anonymising proxies and VPNs operate which will help in the development of solutions to detect the usage of both. The literature review gives descriptions of various techniques used in general network Intrusion Detection Systems, including the integration of machine learning techniques into Intrusion Detection systems. It also provides a review of machine learning techniques with a more in depth look into Neural Networks. Looking at the previous research conducted on network traffic classification, it can be seen that there has been a lot of success in classifying traffic using Multi-layered Neural Networks that are trained on the network packet data and also trained on time-based TCP flow statistics. Based on this information a hypothesis was formed that a Multi-layered Neural Network trained on TCP packet data would be capable of classifying network traffic as originating from an Anonymising Proxy/VPN or from a non-anonymising source.

3. Detection of Anonymising Proxies

This chapter aims to develop a strategy for distinguishing between TCP network packets that originate from a proxy server and packets which do not. The model developed in this implementation chapter is based on a Multi-layered perceptron neural network that is trained on captured network packets. Section 3.1 provides an overview of the hardware & software environment, Section 3.2 details the capture of the required types of network packet and their subsequent compilation into training and testing datasets. Section 3.3 provides a brief overview of the Azure machine learning studio upon which the experiments were run.

3.1 Introduction

The client machine used to initiate connections and send requests through the web proxies is a virtual machine (VM) hosted using the desktop virtualisation software VirtualBox[6]. The host system used to run the VM is equipped with a quad core Intel i7 processor and 24GB of DDR3 RAM. The VM has access to 4 threads from the processor and 6GB of RAM. The operating system chosen for the VM was Ubuntu 16.04 and this was later upgraded to 17.10. A Linux operating system was chosen because of the ease of automation for the capture of data and then packaging it into a suitable format. It was also a preferred choice due to the ease of programming with python using the built-in terminal command prompt. For capturing the network data, the VirtualBox network interfaces needed to be set up. VirtualBox provides up to eight virtual PCI Ethernet cards for each virtual machine. For each card, the individual hardware that is virtualised and the mode in which it is virtualised can be selected, with respect to the physical interface on the host machine. Each of the virtual network hardware types represents a different physical hardware PCI Ethernet card, with each card having different compatibilities with various operating systems. For the purposes of capturing network data from an Ubuntu VM, the Intel PRO/1000 MT Desktop virtual network card was left as the default choice. Each network adapter can also be configured to operate in a different mode. The mode selected for capturing the network data was the Bridged Networking mode. When this mode is enabled, the VM connects

[6] https://www.virtualbox.org/

directly to the host machines network card and exchanged packets directly, circumventing the host operating systems network stack.

3.2 Dataset

When training neural networks, a dataset containing example data is required. The data included in the dataset can either be labelled or unlabelled, but for the purposes of training a neural network, which is a supervised learning model, the data needs to be labelled. For the purposes of these experiments, there are two labels. Data generated from anonymising web proxies is given a label of '*1'* and traffic that is not generated from the proxies is given a label of *'0'*.

The first step in compiling training and testing datasets is gathering the actual raw data. The data being used for these experiments will be in the form of Transmission Control Protocol (TCP) network packets excluding the payload section. Figure 3.1 shows a representation of a TCP header, giving an overview of what is transmitted by the protocol. The choice to exclude the payload of the TCP packet was made after researching methods to decrypt packets in order to scan their contents and finding that, whilst there are methods available to accomplish this, they have their own security risks as they involve man in the middle (MiTM) style attacks which could expose sensitive information such as encryption certificates.

Figure 3.1 shows the fields of the TCP header and a short description of what the purpose of the field is. Also included in the data will be fields from the IP header. However, these fields are for organising the data and won't be included in the neural network training. This is to ensure that the neural network does not overfit the data by focusing on the IP addresses of the web proxies.

TCP Header

Offsets Octet		0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port														
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0				NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size														
16	128	Checksum																	Urgent pointer (if URG set)													
20 ...	160 ...	Options (if *data offset* > 5. Padded at the end with "0" bytes if necessary.) ...																															

Figure 3.1: TCP Header

TCP Field	Usage
Source Port	The Source Port is the port number used by the computer sending the TCP segment and is usually a number above 1024 (but not always).
Destination Port	The Destination Port is the port number used by the computer receiving the TCP packet and is usually a number below 1024 (but not always).
Sequence Number	The sequence number helps the TCP software on both sides keep track of how much data has been transferred and to put the data back into the correct order if it is received in the wrong order, and to request data when it has been lost in transit.
Acknowledgement number	The acknowledgement number acknowledges receipt of data that has been received, if any. It also indicates the value of the next sequence number that the receiver is expecting.
Data Offset	Specifies the size of the TCP header in 32-bit words
Reserved	Set aside for future use and should be zero
URG	**Urgent Flag:** Used to indicate if "urgent" data is contained in the packet
ACK	**Acknowledgement Flag**: Used during 3-way handshake and data transfers.
PSH	**Push Flag**: Used for TCP push, which returns the buffer to the user application. Used primarily in streaming.
RST	**Reset Flag:** Used to reset a TCP connection
SYN	**Synchronise Flag:** Used during 3-way handshake
FIN	Indicates end of the TCP session
Window	Number of octets in the TCP header
Checksum	This field is used by the receiver to verify the integrity of the data in the TCP payload and rejects data that fails the CRC check.
Urgent Pointer	Points to the end of "urgent" data in the packet, but this field only exists if the URG flag is set.
Options	Used to indicate the options used, if any.
Padding	Used to ensure that the TCP header ends on a 32-bit boundary.
Data	This field contains a segment of data from the user application, such as part of an email or web page.

Table 3.1: List of TPC fields and description of their function

3.2.1 Packet capture

To gather network packets that originated from a proxy service for use with the dataset, a list of such proxy sites needed to be collected. The best source for this was the various "proxy lists" available on the internet. These websites collect together a recent list of known proxy servers that are available for use with an interest in advertising their own services. Some of the lists only show sites that require payment to access the proxy service, some sites list a mix of paid and free proxy services and others list only those that are free from charge. One such list[7] is operated by a company called *UpsideOut* who operate their own proxy service called *Proxify*. At the time of writing, the site is listing 50,824 different web proxies. After a short review, it was discovered that not every site on this list is actually online, however a list of sites that were accessible at the time of the experiments was gathered.

Generating the network traffic required for the dataset involves using the proxy sites to visit websites. Doing this manually would have taken a large amount of time so a solution was developed to automate the browsing. It was decided that the scripting language Python would be used for development. Python has great support for working with networks and automation of functions, which is exactly what is required to generate this dataset. Familiarity with the language also played a part as there are other languages which are useful for the purposes of handling data, such as R, but would have taken time to learn. The python library selenium includes a package called *Splinter* which allows a python script to interact with an installed web browser. Splinter allows a python script to interact with elements contained within the HTML code of websites, such as filling out text fields or clicking buttons. URL addresses are provided in the form of a string containing the full address, for example: "https://www.website.com/". Figure 3.2 shows a flowchart which describes operation of the automated browsing script. Figure 3.4 goes on to elaborate on the packet capture script that is used to capture, process and record the network traffic for the dataset. To create a script capable of browsing multiple sites, a string array can be used, as shown in Figure 3.3. A selection of the web proxy sites used is also shown.

[7] https://proxy.org/web_proxies.shtml

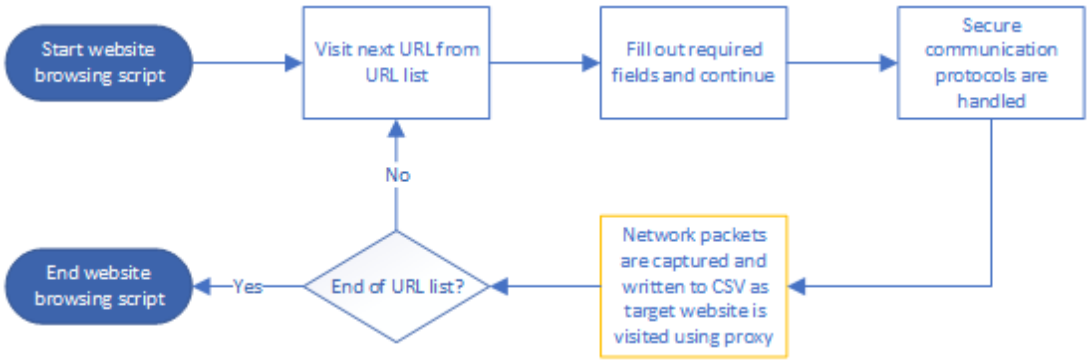

Figure 3.2: Flowchart describing automatic browsing of proxy websites.

```
url = [
        "https://secure.cogsoz.com/proxy/",
        "http://samstevenm.net/prox/", "https://muadness.com/proxy/",
        "http://www.radiocarb.com/p/", "http://proxy.rimmer.su/",
        "https://awssl.com/", "https://moka4.com/",
        "https://webproxy.stealthy.co/", "http://bvpn.win/", "http://www.emuby.com/",
        "http://www.docoja.com/blue/index.php"]
```

Figure 3.3: Code snippet for URL string array.

The code used to browse to and interact with the sites shown in Figure 3.3 can be seen in figure 3.4. The first thing that is accomplished is actually browsing to the site. What site is being browsed to is determined by the for loops position in the string array. The script is then instructed to sleep for two seconds to allow the site to fully load before moving on to the next part. This next part finds the text input field of the proxy server by its CSS id, which was determined to be “input” on most of the Glype, PHP and CGI proxy servers. The site that is the target site for the proxies is “www.whatismyipaddress.com”. This is a website that displays the connecting machine’s IP address, which in the case of the script would be the IP address of the proxy server that is acting as an intermediator.

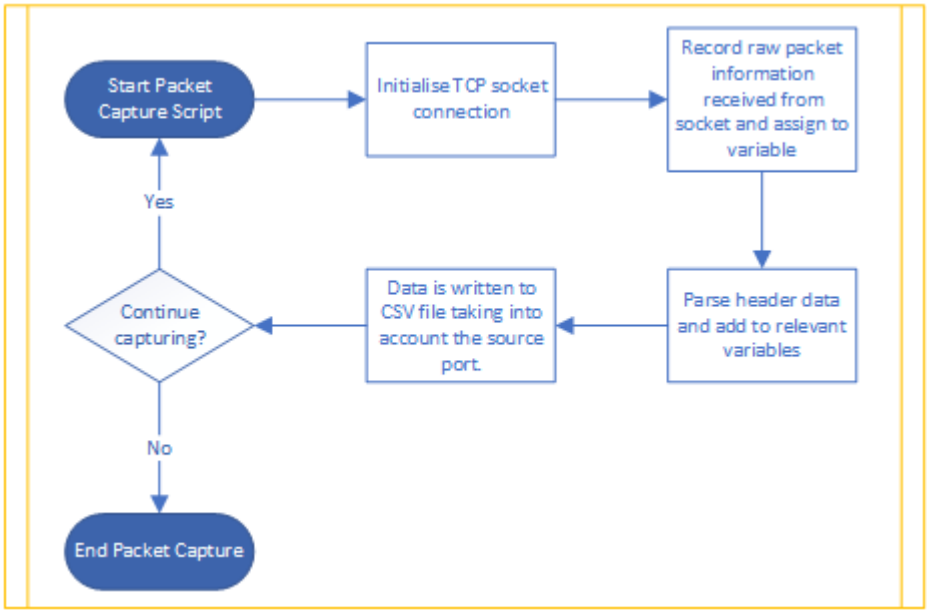

Figure 3.4: Flowchart describing the operation of the packet capture script

```
for site in url:
    # Visit the site using Browser
    b.visit(site)
    time.sleep(2)
    # Find and fill the textbox then find the submit button and 'click' it
    b.find_by_id('input').fill('www.whatismyipaddress.com')
    time.sleep(1)
    if b.is_element_present_by_css('input.button'):
        goButton = b.find_by_css('input.button')
        goButton.click()
    elif b.is_element_present_by_css('input.submitbutton'):
        goButton = b.find_by_css('input.submitbutton')
        goButton.click()
    time.sleep(2)

    # Deal with SSL warning page if it appears
    sslWarningPage = b.find_by_text('Warning!')
    if sslWarningPage is not None:
        print('Warning encountered, dealing with it...')

        continueButton = b.find_by_css('input')[1]
        continueButton.click()

b.quit
```

Figure 3.5: Code snippet for browsing to and interacting with proxy sites.

The script is then instructed to wait for one second before finding and clicking the submit button which initiates the connection to the target site. Some of the proxy sites used do not support the use of SSL encryption, so when browsing to a website that does make use of encryption, which the target site does, the proxy will display a warning page informing the user that the connection will not be encrypted. Below the

warning, a button is provided which allows the user to continue on despite the lack of security. The script checks for this warning and if it is found, it proceeds to locate the continue button and click it. The full browsing script can be found in appendix A.

```
#create an INET, streaming socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
#connect to web server on TCP port 80
s.connect((“www.examplewebsite.com”, 80))
```

Figure 3.6: Example browser socket connection

The capture of the packets and compilation into a Comma Separated Value (CSV) dataset is also handled with a python script. There are various packet capture tools available. The script makes use of python’s socket library which allows for the programming of various sockets. Sockets are one of the most popular methods of inter process communication and are used extensively in network communications. Browsers make use of sockets whenever they attempt to connect to a website. An example browser connection is shown in figure 3.6.

```
try:
    s = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_TCP)
except socket.error:
    print('Socket could not be created.')
    sys.exit()
```

Figure 3.7: Creation of TCP capture socket.

For the purposes of data capture, a socket can be opened specifying that the socket listen for every raw TCP packet that arrives at the machines network card.

```
while True:
    packet = s.recvfrom(65565)
    # Transfer tuple contents to string type.
    packet = packet[0]
    # Take first 20 bytes for the ip header.
    # Ethernet header is usually before, but we aren't capturing that.
    ip_header = packet[0:20]
    # Unpack from bytes format
    iph = unpack('!BBHHHBBH4s4s', ip_header)
    version_ihl = iph[0]
    version = version_ihl >> 4
    ihl = version_ihl & 0xF
    iph_length = ihl * 4
    ttl = iph[5]
    protocol = iph[6]
    s_addr = socket.inet_ntoa(iph[8])
    d_addr = socket.inet_ntoa(iph[9])
```

Figure 3.8: IP header extraction

Figure 3.8 shows how the socket is set up in the python packet capture script. To receive the packet data and assign it to a variable the line *packet = s.recvfrom(65565)* is used. This assigned the raw packet data to a variable which can then be used to extract details from the packet. Normally the ethernet header would also be captured, but because of the way the socket is setup, the ethernet header is omitted from the captured data. The IP header of the packet takes up the first 20 bytes, so the variable *ip_header* is assigned that data, as can be seen in figure 3.8. Figure 3.9 shows the extraction of the TCP header. For both headers, the data is unpacked from the bytes format, which allows for the accurate placement of individual sections of the header to the appropriate variables.

```
# TCP header starts right after IP header and is usually
# 20 bytes long
tcp_header = packet[20:40]
# Unpack from bytes format
tcph = unpack('!HHLLBBHHH', tcp_header)
source_port = tcph[0]
dest_port = tcph[1]
sequence = tcph[2]
acknowledgement = tcph[3]
doff_reserved = tcph[4]
tcph_length = doff_reserved >> 4
h_size = iph_length + tcph_length * 4
data_size = len(packet) - h_size
```

Figure 3.9: TCP header extraction

Following the extraction of the header details, the script then determines where the TCP flags are stored and assigns those bits to a variable. An if statement compares the value stored in the variable to a known list of hexadecimal values that represent each TCP flag and then assigns the value of the found flag to a string variable. To handle transferring the captured network data to a format suitable for storing and working with datasets, the script uses the “csv” import option to create a writer object. The code used is shown in figure 3.10.

```
outputFile = open('vpntraffictest.csv', 'w', newline='')
writer = csv.writer(outputFile)
# Write out the top row
writer.writerow(['Version', 'Protocol', 'TTL', 'SrcAddr', 'DestAddr',
                 'SrcPort', 'DestPort', 'SeqNum', 'AckNum', 'Flag', 'dataSize',
                 'Service', 'Label'])
```

Figure 3.10: Opening csv writer object

First the destination file for the data is defined, in this case it is “vpntraffictest.csv”. This creates a Comma Separated Value (CSV) file in the same directory that the packet capture script is stored in. This file needs to be created before the writer object is created so that the writer knows where its output destination is otherwise an error will occur. The writer is created by calling the writer from the CSV library and passing the *outputFile* variable to it. Before the packet capture begins, the CSV writer writes the first row of the dataset which are the names of each part of the TCP packets that are being captured.

```
            if source_port == 80:
                writer.writerow([str(version), str(protocol),
                                 str(ttl), str(s_addr),
                                 str(d_addr), str(source_port),
                                 str(dest_port), str(sequence),
                                 str(acknowledgement), Flag,
                                 str(data_size), "HTTP", "0"])
                print("Packet Captured")
            else:
                writer.writerow([str(version), str(protocol),
                                 str(ttl), str(s_addr),
                                 str(d_addr), str(source_port),
                                 str(dest_port), str(sequence),
                                 str(acknowledgement), Flag,
                                 str(data_size), "HTTPS", "0"])
                print("Packet Captured")
    outputFile.close()
```

Figure 3.11: Filtering traffic to only write HTTP and HTTPS packets

Once a packet has been captured and processed by the script, the details of the packet are written to the output file. The traffic that is being investigated is Hypertext Transfer Protocol (HTTP) or Hypertext Transfer Protocol Secure (HTTPS) web traffic, however the packet capture script captures all TCP traffic. To strip out the unwanted traffic and only record the HTTP/HTTPS traffic, the TCP source port is used. Figure 3.11 shows an if statement which instructs the CSV writer to only write the details of the TCP header to the output file if the source port is either port 80 (representing HTTP traffic) or port 443 (representing HTTPS).

3.2.2 Non-proxy data capture

Training a binary class classification algorithm requires two different classes of data to be provided. As a comparison to the proxy network packet data, packets were captured from the same system without the use of a proxy. To do this, the automated web browsing python script which first visited a proxy site then used the proxy to browse to "*whatismyipaddress.com*" was modified as shown in figure 3.12.

```
from time import sleep
from splinter import Browser
from random import randint

b = Browser()
url = ["https://www.theregister.co.uk/", "https://arstechnica.co.uk/",
       "http://www.bbc.co.uk", "https://news.ycombinator.com/",
       "http://whatismyipaddress.com/", "https://www.google.com",
       "https://www.youtube.com", "https://www.facebook.com",
       "https://www.wikipedia.org", "https://www.yahoo.com",
       "https://www.reddit.com", "https://www.amazon.com",
       "https://www.live.com", "https://www.instagram.com",
       "https://www.linkedin.com", "https://www.netflix.com"]

while True:
    for site in url:
        b.visit(site)
```

Figure 3.12: Non-proxy target websites

The code instructing the browser to visit the proxy site were removed. The URL list was populated with a variety of sites from the Alexa top 500 index[8]. The script then instructs the browser to visit each of the sites repeatedly until the execution is manually cancelled. The same packet capture script that is used to capture the proxy network packets is also used to capture the non-proxy data and write it out to a CSV file.

3.3 Experiments

There are a few different technologies and platforms that can be used to conduct the experiments. These are the cloud computing platforms (in the form of Machine Learning as a Service (MLaaS)) of AWS[9], Google Cloud[10], IBM[11], Azure[12] and the local platforms Scikit-Learn and MATLAB. Local platforms have a range of benefits that leverage the underlying hardware of the local machine however this has the undesired effect of being dependent on the hardware available. This dependency lead to the development of distributed techniques which later evolved into what is known today as cloud computing.

Cloud computing is defined as a general computing model for enabling convenient, on-demand access to a shared pool of configurable, distributed computing resources that can be quickly and efficiently provisioned with minimal oversight or interaction

[8] https://www.alexa.com/topsites
[9] https://aws.amazon.com/machine-learning/#
[10] https://cloud.google.com/ml-engine/
[11] https://www.ibm.com/uk-en/cloud/machine-learning
[12] https://azure.microsoft.com/en-gb/services/machine-learning-studio/

with the service provider (Mell & Grance, 2011). With the rise in popularity of machine learning in research, cloud computing providers have fully integrated machine learning into their cloud platforms. A key advantage that cloud computing provides for this research was the removal of hardware concerns and enabled the pursuit of more accurate results.

Amazon's machine learning service (Amazon ML) is designed for users who have no previous knowledge of machine learning. This can be a key advantage for enterprise users, but in the context of this research it is quite limiting in what can be explored using the platform. For example, because the user is not required to have any knowledge of machine learning techniques, the service restricts the choice of the machine learning method used to one of the platforms choosing through analysis of the data provided.

Google Cloud's machine learning service resembled Amazon ML in that it was aimed at novice users and restricted the choice of machine learning algorithms even for machine learning engineers. The Google service is based around the TensorFlow suite which is a library maintained by Google that has the reputation being powerful but is accompanied by a steep learning curve and is designed for machine learning tasks which rely on specific neural network architectures.

Azure's machine learning service (ML Studio) was found to provide machine learning tools to both novices and experienced data scientists. In comparison to both Amazon and Google, Azure offers a larger variety of algorithms to accomplish machine learning tasks and doesn't restrict the choice of which algorithm is used. Furthermore, it allows for the creation of original machine learning algorithms and is not restricted to out-of-the-box algorithms. This advantage alone makes Azure the obvious choice to pursue in the following experiments because of the need for highly accurate results without the restriction to algorithm choice.

Azure Machine Learning studio is a cloud service that provides an IDE-like workspace to allow for easier building, testing and deployment of predictive analytic models. Models can be constructed by dragging and dropping dataset and analysis modules into a workspace area. Modules can be added iteratively to help pinpoint problems. Predictive analysis helps you predict what will happen in the future. It is used to predict the probability of an uncertain outcome. Azure offers various types of

statistical and machine learning algorithms aimed at predictive analysis such as neural networks, boosted decision trees and linear regression. Azure outlines a 5-step process to building an experimental predictive model: gather raw data, pre-process the data to clear the data of missing values or other mistakes, define the features that the model will be trained on, choose and train a learning algorithm, test the algorithm (Fontama et al., 2014). Once the model is trained and is predicting the test data accurately it can be deployed as a web service. Azure replaces the original dataset with a module to allow input from the web. Using the C#, python or R programming languages in conjunction with the URL of the deployed web service and a generated key, data can be sent to the web service to be analysed.

The features offered by Azure Machine Learning studio were a large reason that Azure was chosen to conduct the experiments in this chapter. Another big factor in the decision was the ease of use that the IDE style workspace and drag and drop construction offered. This allowed for quick and easy re-configuration of experiments to try out different algorithms and methods for parameter tuning.

3.3.1 Methodology

There have been a number of recent studies that have made use of Azure's machine learning studio. (Bihis & Roychowdhury, 2015) proposed a generalised flow within Azure that would accept multi-class and binary classification datasets and process them to maximise the overall classification accuracy. Two sets of experiments were run. The first was to benchmark the Azure machine learning platform using three public datasets. The second was to evaluate the proposed generalised flow using a generated multi-class dataset. The results showed that the generalised flow improved accuracy in all but one of the comparisons with prior work.

(Pathak et al., 2015) describes a methodology to obtain a real-time view of traffic issues in a city using status updates, tweets and comments on social media networks using state of the art machine learning. The machine learning capability was provided by Azure machine learning studio. Data from various social networks is polled continuously by a worker role process hosted in Azure. The machine learning studio

is used to process the data and analyse the text being imported. For the experiment they annotated 1100 social network feeds for 4 cities. This data was then split into training, cross validation and testing datasets that were used to train and test the machine learning algorithms in Azure machine learning studio. Classification accuracy for one social network ranged from 90-95% whereas on another the accuracy was just higher than 75%. (Krithika & Narayanan, 2015) proposed a method aimed at grading short test answers using machine learning techniques implemented in Azure. Experiments were run using 152 samples of student answers to a computer science question. The experiment showed that the system was able to grade all of the answers correctly after testing. (Tselykh & Petukhov, 2015) proposed an anti-fraud web service that employed machine learning algorithms for predictive analytics in order to reduce the costs of infrastructure and software. Azure machine learning studio was used to provide the machine learning aspect. When building the machine learning model in Azure, they experimented with several algorithms for two-class classification. Using Azure's built in Score Model module, they were able to achieve an accuracy of 88% and went on to publish the model as a web service that was capable of performing anti-fraud activities whilst reducing the cost of such a service to virtually zero.

3.3.2 Two-Class Neural Network

The algorithm that was selected for classification of proxy network traffic was the Azure module "Two-Class Neural Network". There is also a module provided for a multiple class application however that doesn't apply for this task.

The decision to use a Multi-layered Neural Network was based both on the review of prior literature on network traffic classification and on early test experiments run using a selection of the other machine learning algorithms including Bayes Point Machine, Support Vector Machine, Logistic Regression, Decision Forest and Boosted Decision Tree. These early experiments showed that the Two-Class Neural Network module provided the best base for improvement when compared to the other results. Other machine learning techniques such as Fuzzy Logic or Genetic Algorithms were ruled out as they were found to be inappropriate for the problem of classifying network traffic based on the packet data. Classical logic only permits results which are either

true or false. However, with some problems there is the potential for the result to be within a range between completely true or completely false. Fuzzy logic is the solution for such problems, but for the classification of network traffic as belonging to an Anonymising Proxy or not the result is a simple binary which Fuzzy logic is unsuited for. In addition, it was found that a genetic algorithm was unnecessarily complex for this experiment, although it may have a place in some future work which will be addressed in Chapter 5.

Figure 3.13 shows how this module appears in the Azure machine learning studio interface. To aid the configuration of the different algorithms that Azure has Microsoft provides an extensive documentation resource.

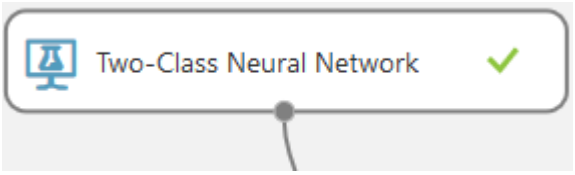

Figure 3.13: Azure Neural Network module

The documentation for the two-class neural network module offers information on how to configure the parameters of the algorithm for two scenarios; whenever the parameter configuration is already known and when the optimal parameters are still unknown.[13]

Figure 3.14 is a screen capture of the parameter options available for the two-class neural network module. The parameters that can be seen starts with the trainer mode. The trainer mode is the parameter that sets the algorithm up for one of the two scenarios that were mentioned. It contains two options, “Single Parameter” and “Parameter Range”. The Single Parameter option allows the user to enter a single value for each of the parameters whereas the Parameter Range option allows for multiple value ranges to be used. The latter was the selected option for the proxy classification problem as the optimal parameter values were unknown beforehand. This is then combined with the module “Tune model hyperparameters” module which performs a parameter sweep over the specified settings and learns an optimal set of hyperparameters. This process is referred to as “tuning”.

[13] https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/two-class-neural-network

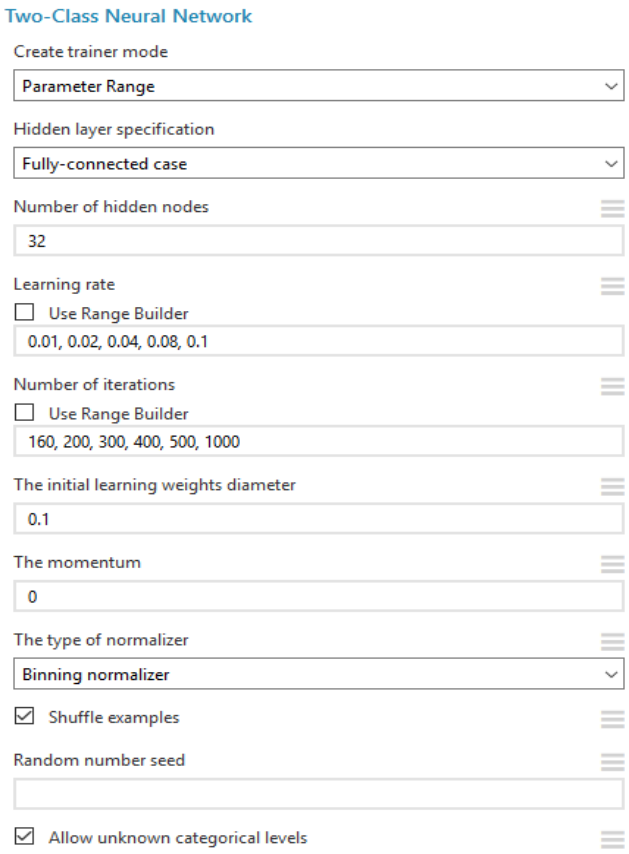

Figure 3.14: Azure neural network parameters

Figure 3.14 shows the ranges chosen for both the learning rate and the number of training iterations of the model. When the model is being tuned, these ranges instruct the tuning module what parameter values should be used during the parameter sweep. The specification of the hidden layer can either be a fully connected instance, as selected, or it can be defined using a custom script written in the Net# language. The default, fully connected case uses a pre-defined specification for the hidden layer. This results in a neural network which has one hidden layer, an output layer that is fully connected to the hidden layer which is in turn fully connected to the input layer. The number of nodes in the input layer equals the number of features used in the training data and the number of nodes in the hidden layer is defined by the user in the parameter option.

The number of nodes in the output layer equals the number of classes, which for a two-class network means that all inputs will map to one of two nodes. The custom

definition script is useful for when a more complex network is required, such as when deep learning is being implemented.

3.3.3 Dataset upload and preparation

To use a dataset with Azure it first needs to be uploaded to the machine learning studio. Azure supports multiple dataset types including CSV files. Before uploading the entire dataset samples randomly removed which were then used to compile a separate testing dataset. This left a training dataset of 5954 samples and a testing set of 1002 samples which were to be kept separate from the training process. Once uploaded, the datasets can then be added to the Azure interface and can then be further prepared for training and testing the neural network. The first preparation steps are in adjusting the metadata of the dataset so the model understands what fields are training features and what are class labels in the data. This is metadata that would not be readily available as part of the CSV format. The metadata is adjusted using the "Edit Metadata" module. In this module, the field that is to be edited is first selected and then there are four changes that can be made. The first deals with the data type of the field, for example, string or integer. If the field is already defined as being the data type required or simply does not need a type associated with it, there is the option to leave it unchanged. This applies to the first three modifications available. The second defines the field as either being a categorical field or not. If the field is already as required there is the option to leave it unchanged. The third modification allows the user to change whether the field contains a feature, a label or a weight. Again, there is the option to leave it unchanged and there are also options available to clear a previous definition. The fourth and final modification allows the user to enter new column names for the selected fields.

For feature selection, Azure machine learning studio provides three modules: "Filter Based Feature Selection", "Fisher Linear Discriminant Analysis" and "Permutation Feature Importance". For the purposes of this experiment however, feature selection was performed manually as there were only twelve features to select from. Some features were removed from the training dataset because they did not offer any value to the training. These were the Version number, Protocol identifier and the Time To

Live (TTL). These fields contained the same value for each row of the dataset and therefore would have affected the computation time with no benefit gained.

The source and destination IP addresses were removed as these could very well cause the network to overfit to the problem during training as the Neural Network would associate the IP address values with the training label. If the network focuses too much on the IP addresses then the system would operate like an over-complicated IP blacklist rather than an intelligent system. From there, experimentation was used to ascertain what features would cause the network to either overperform or underperform. This involved the systematic removal of features and observing what effect this had on the training of the network. Whilst this was a slower technique, this was offset again by the small number of features and also provided information on the effect each feature had on the training allowing for more informed choices to be made. For example, through this method it was found that the feature that had the most influence over the training was the TCP Acknowledgement number whilst the feature that seemed to have the least influence was the feature that noted whether HTTP or HTTPS were used. The module used to select and remove features from the training is the “Select Columns in Dataset” module. All of the desired training features plus the classification label are selected using this module. The features that were used to train the network were: Source Port number, Destination Port number, TCP Sequence number, TCP Acknowledgement number, TCP Flag, Data size in bytes and whether HTTPS or HTTP were used. To eliminate underperformance of the model, the hyper-parameter tuning mentioned at the end of section 3.3.2 was used.

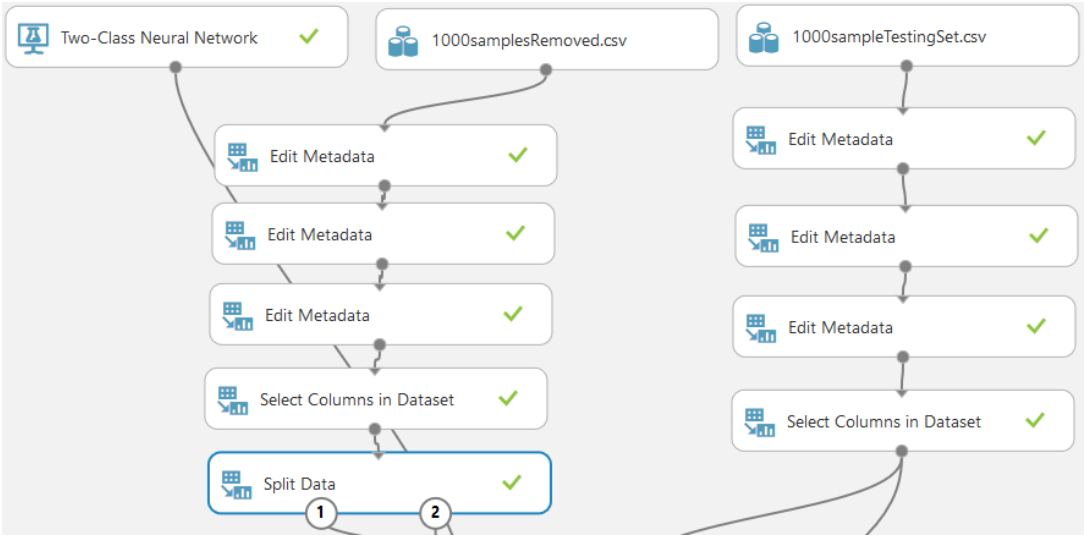

Figure 3.15: Dataset preparation

The final step before training and tuning commenced was to split the training dataset to provide both a training dataset and a training validation dataset. This is accomplished using the "Split Data" module and an 80/20 split was used which resulted in a final training dataset of 4763 and a validation set of 1191. Figure 3.15 shows the dataset preparation as it is represented in Azure. Also shown on the right-hand side is the 1002 sample testing dataset which is also going through the same preparation as the training dataset. This is so it can be successfully tested against as otherwise Azure will throw an error.

3.3.4 Training and Testing

Figure 3.16 shows the training and testing portion of the experiment as it is represented on the Azure machine learning studio.

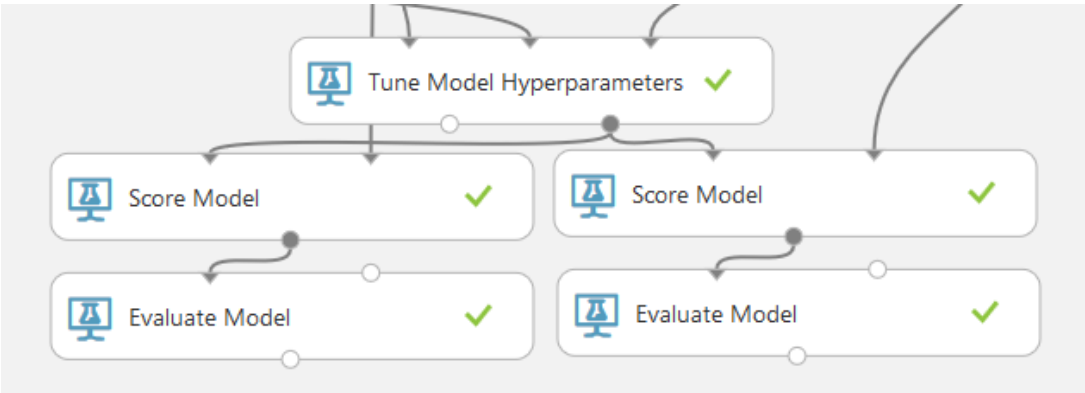

Figure 3.16: Training and testing of the neural network

Figure 3.17 shows the entire experiment. Once the data has been fully prepared, it is connected to the tuning module as the training dataset. The two-class neural network model is also connected to the tuning module at this time. At this point the experiment can be run with the tuning module's default settings, however these settings were modified slightly. By default, the tuning module will only perform 5 parameter sweeps.

This was changed to 50 sweeps, so the final model could be as close to the best model as possible. Training time for this model lasted for approximately one hour. This could be considered a consequence of the use of Azure, specifically the use of the free workspace, as there was no control over what hardware was used in order to train the model. The studio workspace used is in the South-Central US region which could also add latency to the connection, further slowing the process. It is possible to purchase a subscription to Azure which unlocks a far greater feature set. This was deemed unnecessary for this experiment as the resources provided were enough.

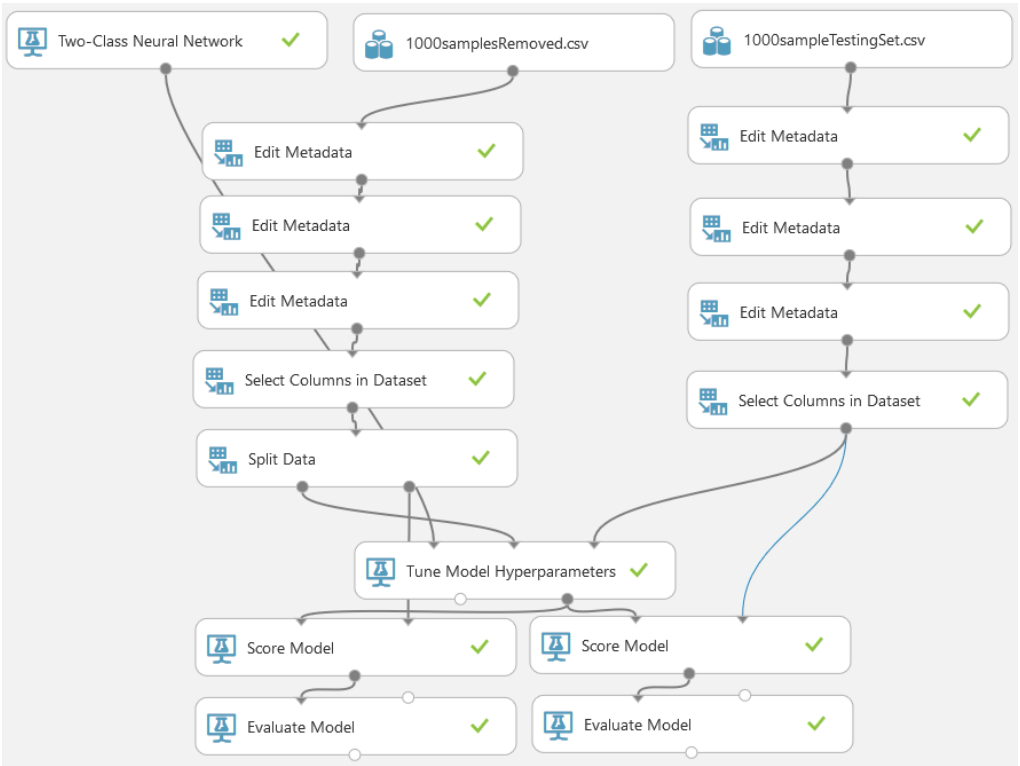

Figure 3.17: Fully connected experiment

Once tuning is finished, the tuning module outputs a trained version of the best model it found. For this experiment, the best model used a learning rate of 0.0441313, a Squared Error loss function and was trained over 433 iterations. To score and evaluate the model based on the validation dataset and then the separate training dataset, the "Score" and "Evaluate" modules are used sequentially.

The score module scores the classification predictions for the trained model and the outputs those results as a scored dataset. The scored dataset is then passed to the evaluate module which calculates a range of metrics based on the results. These metrics include: Accuracy, Precision, Recall, F-score, Area Under Curve (AUC), Average Log Loss and Training Log Loss.

3.3.5 Results

The results gathered from the separate testing set are shown in figure 3.18 along with the confusion matrix. The ROC curve is shown in figure 3.19. The AUC for the test was 0.988

True Positive	False Negative	Accuracy	Precision
460	41	0.946	0.973
False Positive	**True Negative**	**Recall**	**F1 Score**
13	488	0.918	0.945

Figure 3.18: Confusion Matrix and Results

The results in figure 3.18 show an overall classification accuracy of 94.6% for 1002 samples. There were only 13 false positive classifications, which is relatively low. However, there were 41 false negatives which is quite high. This may be due to the higher amount of negative (i.e. non-proxy) samples than positive samples at 473 positives to 529 negatives. In the context of this model being used as a proxy detection system on a live network, 41 proxy packets in every 1002 (0.04%) packets would possibly avoid detection and 13 in every 1002 (0.013%) normal, non-proxy packets would be possibly detected erroneously.

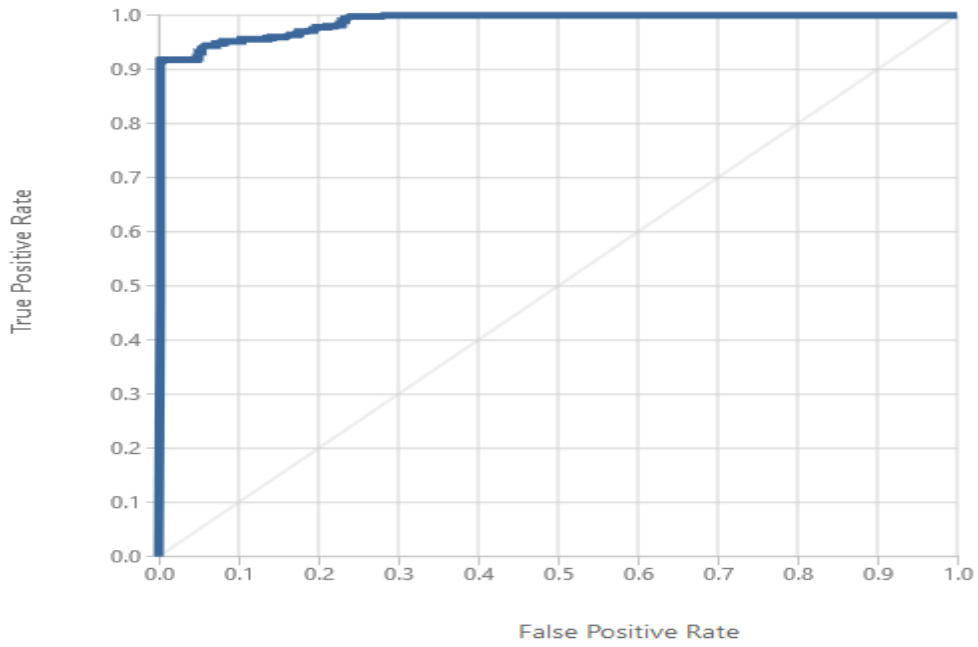

Figure 3.19: ROC Curve

Overall there is the possibility for approximately 50 errors in every 1000 packets. Whilst that may warrant worry, it is something that could possibly be improved upon and overall the results seem be an indication that the model has the capability of detecting proxy network packets.

A drawback of this approach is that the dataset used is relatively small at approximately 7000 samples. Unfortunately, due to the nature of the packet capture and the tools available, the capture was time consuming. Another potential drawback is the use of TCP header details as training features. As there are only 12 features total, which are then reduced to 7, there is not much leeway given to the possibility that some features may not be suitable for classification at times, depending on the network conditions. Some solutions to these problems are discussed in section 5, future work.

3.4 Summary

This chapter has explored the development of a neural network model capable of classifying TCP network packets as either web proxy traffic or not using the data contained within the TCP header. Section 3.2 describes the capture of network packets from both proxy and non-proxy sources with details given about the python scripts

created. Section 3.3 describes the client machine used to generate the network traffic captured. Section 3.4 then gives an overview of the Azure machine learning studio with examples of research conducted with it. The section follows on with details about the experiment conducted in this thesis using the machine learning studio. Feature selection, setup of the neural network model and how the model was trained are all discussed. Finally, the results from the experiment are described.

4. VPN classification

4.1 Introduction

In the previous chapter, classification of network traffic that was being passed through a web-proxy was investigated using back-propagation artificial networks with a focus on using the TCP header details from captured network traffic as the dataset. This chapter aims to investigate the classification of VPNs using a similar neural network, albeit this time not using the cloud machine learning service from Azure.

Virtual Private Networks (VPNs) are quickly becoming a popular method for criminals and other bad actors to hide their online activities (Harmening, 2013). This is helped along by the increase in ease of use of VPNs; they are no longer just a tool for remotely accessing enterprise resources when travelling for work or when working from home. In fact, this could be a use-case for a criminal. If they wish to remotely access an enterprise network in order to steal company and trade secrets, they can use a VPN (or multiple VPNs) in order to hide their own location or to make it appear as if someone else was infiltrating the network (Geetha & Phamila, 2016). There have been a few notable cases of this happening in recent years, such as the Sony Pictures incident from 2014, where confidential data including personal information about employees was stolen (Peterson, 2014). It is likely that the attackers used a VPN to hide their location and identity as, to this date, no one has been officially charged with the crime and brought in front of a court (Pagliery, 2014).

Other attacks of note are the various data breaches which have been occurring for the last number of years, such as the LinkedIn breach of 2012 which was only discovered in 2016 (Hunt, 2016). Approximately 167 million account details including emails and passwords were stolen. It is not known whether the attacker(s) were using a VPN service to hide their location. The ability to detect whether a VPN has been used or not could be helpful in the pursuit of attackers such as those just mentioned.

The classification method described in this chapter is envisaged as a step towards the creation of a VPN detection framework that could be used to help law enforcement officials track down those responsible for attacks.

4.2 Dataset

The dataset for the previous chapter consisted of TCP packet data. The results indicated that this dataset provided enough data to train a Neural Network to recognise the different patterns between traffic that was using a proxy and traffic that was not. Moving forward from this work, it was decided to attempt a similar approach in order to train a Neural Network to classify VPN based traffic. A dataset consisting of TCP packets captured using the packet analysis tool Wireshark from an OpenVPN connection was created and tested using the exact same Azure machine learning tools. The results for this showed that the network was overfitting the problem as it was achieving 100% classification accuracy for both VPN traffic and non-VPN traffic. In external validation tests, the network was essentially guessing, as it was classifying every sample as having come from a VPN. In order to overcome this problem, a new dataset consisting of TCP flow records/statistics was proposed as more appropriate for analysis. This decision to form a new dataset was inspired by previous works of (Draper-Gil et al., 2016; Stevanovic & Pedersen, 2014; Soysal & Schmidt, 2010), with particular note to the work of (Draper-Gil et al., 2016) which presented a flow-based classification model to classify encrypted and VPN traffic using only time-related features. Flow statistics provide a high-level view of network communications by reporting the addresses, ports and byte and packet counts contained in those communications (Cisco, 2018). This data can be especially valuable when network traffic is being encrypted which can be the case with VPN traffic. More detail is provided regarding the production of the flow records used for this piece of work in section 4.2.2.

4.2.1 Capture Method

Wireshark formed the basis of the packet capture for this newer dataset as was also the case for the first dataset. The computer system used to capture the traffic was an Ubuntu 16.04 based virtual machine running on a Windows 10 host. The network connection used in the experiment is a virtualised Intel PRO gigabit ethernet card

although this shouldn't have any effect on the results. While Linux might not be a mainstream operating system for the standard user, it is popular among "bad actors" because of its ability to be heavily customised according to the user's preferences. It allows for a finer degree of control over some of the internal systems included such as networking stack. Figure 4.1 shows a flowchart which describes the entire packet capture process used for building the VPN dataset. Using some built in tools, it is easy to automate connections and disconnections to different networks and different network interfaces. This was a particularly helpful feature when dealing with the capture of VPN based packets. In normal operation, a connection to a VPN starts with a typical TCP "hello" sequence and key exchange. Once the connection is setup, it is only taken down whenever the user stops using the VPN. The connection is one long TCP connection between the user's machine and the VPN server. This created some problems with the NetMate flow statistic calculation tool, which is also discussed in section 4.2.2. The problem with NetMate was solved using a Linux bash shell script and the Linux system's automatic task scheduling tool; *cron*. The shell script contains a one-line command that initiates a connection to the VPN and sets a timeout value of 590 seconds. The full command is shown in figure 4.2.

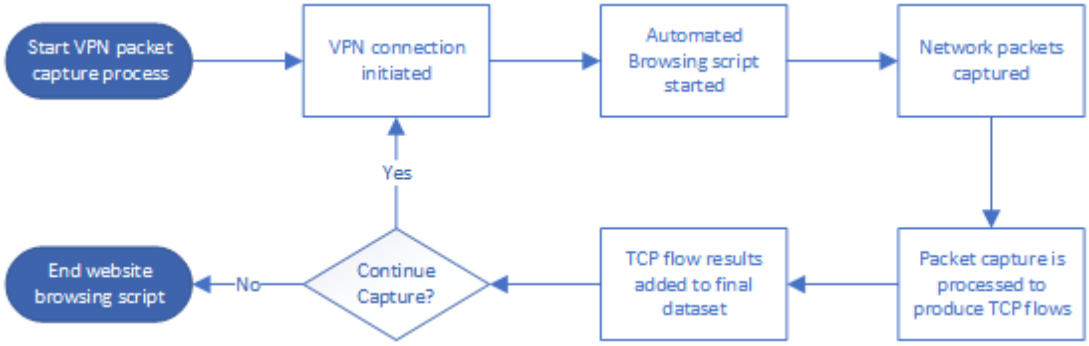

Figure 4.1: Flowchart decribing packet capture process

```
timeout 590s openvpn /etc/openvpn/<openvpn config file>.ovpn
```

Figure 4.2: VPN connection command

The 'timeout' causes the command to quit after the set amount of time and 'openvpn' is the command that will be affected by 'timeout'. Openvpn is a type of VPN server that can be installed easily on many systems. In this command 'openvpn' initiates the connection to the Openvpn server that is described in the config file i.e. <openvpn config file>.ovpn. Combining this command with the automatic scheduling tool *cron*

is straightforward. The goal was to have this command run every ten minutes, so the timeout value is set to 590 seconds which leaves enough time for the connection to completely close. Then, 10 seconds later the command is run once again. This will repeat indefinitely as long as the command is listed in *cron*. To instruct *cron* to run this command as required, the file */etc/crontab* is edited with administrative privileges. A line is added to the file detailing the command to be run and how often it should be run. The line added in order to run the connection shell script every 10 minutes is shown in figure 4.3.

```
# m h dom mon dow user command
*/10 *    * * *    root    sh /path-to-file/connect.sh
```

Figure 4.3: Crontab file example

From left to right, the headings of the crontab file stand for: minutes, hour, day of month, month, week of month, user to run command under and the actual command. This will instruct *cron* to run the shell script every 10 minutes of every day of every month as the user 'root'. With the connection to the VPN automated and refreshing every ten minutes, the next task was to generate the network traffic to be captured by Wireshark. There are tools available to generate random packets based on specific attributes to simulate certain network environments. However, it is much better if the data could be captured from realistic browsing practices. Due to the amount of data that would be required, it would be infeasible for a person to sit and visit websites for 24 hours a day, 7 days a week. A modified version of the automated browsing Python script from the proxy detection work was used in conjunction with a small selection of the most popular Alexa top 500 sites[14]. This script can be seen in figure 4.4.

[14] https://www.alexa.com/topsites

```
from time import sleep
from splinter import Browser
from random import randint

b = Browser()
url = ["https://www.theregister.co.uk/", "https://arstechnica.co.uk/",
       "http://www.bbc.co.uk", "https://news.ycombinator.com/",
       "http://whatismyipaddress.com/", "https://www.google.com",
       "https://www.youtube.com", "https://www.facebook.com",
       "https://www.wikipedia.org", "https://www.yahoo.com",
       "https://www.reddit.com", "https://www.amazon.com",
       "https://www.live.com", "https://www.instagram.com",
       "https://www.linkedin.com", "https://www.netflix.com"]

while True:
    for site in url:
        b.visit(site)
```

Figure 4.4: Modfied browsing script

This ensures that the browsing involves some of the more recent and publicly available sites on the web as well as some of the more popular. For the script to browse sites in a relatively realistic fashion, the sleep method is used in conjunction with the *randint* method. The *sleep* function pauses the execution of the entire script. When combined with *randint*, it is possible to pseudo-randomly set the pause time for each occurrence of *sleep*. With the VPN connecting and disconnecting every 10 minutes, the *randint* method's minimum value was set to 10 seconds and the maximum value set to 300 seconds. This means that the website that is visited by the script will be displayed for a minimum of ten seconds and no longer than 5 minutes. In their paper, (Liu et al., 2010) showed that users judged web pages harshly in the first 10-30 seconds. After this time had passed it was likely that users would spend upwards of 2 minutes on the page. During the 10-minute VPN connection period, the browse script would visit a minimum of 2 web pages.

For complete training of the network, more than one class of data is required so it was necessary to capture network traffic that did not originate from a VPN connection. This capture was accomplished in much the same way as with the capture of the VPN traffic. The browsing script used was exactly the same and Wireshark was again used to capture the traffic. The only difference in this instance was that the automatic connection/disconnection to the VPN was removed. This was simply accomplished by adding a "#" to the start of the Crontab line shown in figure 4.3. This defines the line as a programming comment and is ignored by the Crontab parser. This task can also be accomplished by deleting the line from the file.

4.2.2 NetMate

NetMate is a bidirectional flow exporter and analyser tool used to convert capture files of network traffic into flow records (Haddadi & Zincir-Heywood, 2016). A TCP flow is a sequence of packets between two endpoints as defined by their source IP address and port to a destination IP address and port over a certain length of time (Stibler et al., 1999). A sequence like this will only be considered a flow if it is monitored in both directions. The packets captured from Wireshark meet this requirement, so they are compatible with NetMate. The particular version of NetMate used for this dataset is developed by former NIMS lab member; Daniel Arndt. This version is called Netmate-flowcalc which is a bundle comprising of NetMate v0.9.5 packaged with NetAI modules from v0.1 (Arndt, 2011). The output, if using one of the included rules files, takes the form of a comma separated list of values. Each column corresponds to an attribute or feature of the output. These attributes are defined using an Attribute-Relation File Format (ARFF) header as pictured in figure 4.5.

```
@RELATION <44-flow-features>

@ATTRIBUTE srcip STRING
@ATTRIBUTE srcport NUMERIC
@ATTRIBUTE dstip STRING
@ATTRIBUTE dstport NUMERIC
@ATTRIBUTE proto NUMERIC
@ATTRIBUTE total_fpackets NUMERIC
@ATTRIBUTE total_fvolume NUMERIC
@ATTRIBUTE total_bpackets NUMERIC
@ATTRIBUTE total_bvolume NUMERIC
@ATTRIBUTE min_fpktl NUMERIC
@ATTRIBUTE mean_fpktl NUMERIC
@ATTRIBUTE max_fpktl NUMERIC
@ATTRIBUTE std_fpktl NUMERIC
@ATTRIBUTE min_bpktl NUMERIC
@ATTRIBUTE mean_bpktl NUMERIC
@ATTRIBUTE max_bpktl NUMERIC
@ATTRIBUTE std_bpktl NUMERIC
@ATTRIBUTE min_fiat NUMERIC
@ATTRIBUTE mean_fiat NUMERIC
@ATTRIBUTE max_fiat NUMERIC
@ATTRIBUTE std_fiat NUMERIC
@ATTRIBUTE min_biat NUMERIC
@ATTRIBUTE mean_biat NUMERIC
@ATTRIBUTE max_biat NUMERIC
@ATTRIBUTE std_biat NUMERIC
@ATTRIBUTE duration NUMERIC
@ATTRIBUTE min_active NUMERIC
@ATTRIBUTE mean_active NUMERIC
@ATTRIBUTE max_active NUMERIC
@ATTRIBUTE std_active NUMERIC
@ATTRIBUTE min_idle NUMERIC
@ATTRIBUTE mean_idle NUMERIC
@ATTRIBUTE max_idle NUMERIC
@ATTRIBUTE std_idle NUMERIC
@ATTRIBUTE sflow_fpackets NUMERIC
@ATTRIBUTE sflow_fbytes NUMERIC
@ATTRIBUTE sflow_bpackets NUMERIC
@ATTRIBUTE sflow_bbytes NUMERIC
@ATTRIBUTE fpsh_cnt NUMERIC
@ATTRIBUTE bpsh_cnt NUMERIC
@ATTRIBUTE furg_cnt NUMERIC
@ATTRIBUTE burg_cnt NUMERIC
@ATTRIBUTE total_fhlen NUMERIC
@ATTRIBUTE total_bhlen NUMERIC
```

Figure 4.5: NetMate attributes

The first five attributes on this list are taken directly from the TCP packet header. They include the source IP address and port number; the destination IP address and port number and the protocol being used. The rest of the attributes on this list are flow statistics that are calculated by *NetMate*. The statistics calculated for the majority of the remaining attributes are the minimum, mean, maximum and standard deviation.

Names and descriptions of the full list can be found in table 4.1. These features are similar to those produced by another project named “flowtbag” which contains a more detailed description of the features found in table 1 and can be located on Daniel’s personal GitHub[15].

Attribute Name	Attribute Description
total_fpackets	Totals packets in the forward direction.
total_fvolume	Total bytes in the forward direction.
total_bpackets	Total packets in the backward direction.
total_bvolume	Total bytes in the backward direction.
fpktl	The min, mean, max and standard deviation from the mean of packet sizes in the forward direction.
bpktl	The min, mean, max and standard deviation from the mean of packet sizes in the backward direction.
fiat	The min, mean, max and standard deviation from the mean of time between two packets in the forward direction.
biat	The min, mean, max and standard deviation from the mean of time between two packets in the backward direction.
duration	Total duration of the flow in microseconds.
active	The min, mean, max and standard deviation from the mean of time that the flow was active before going idle.
idle	The min, mean, max and standard deviation from the mean of time that the flow was idle before going active.
sflow_fpackets	Average number of packets in a sub flow in the forward direction.
sflow_fbytes	Average number of bytes in a sub flow in the forward direction.
sflow_bpackets	Average number of packets in a sub flow in the backward direction.
sflow_bbytes	Average number of bytes in a sub flow in the backward direction.
fpsh_cnt	Number of PSH flags set in packets in the forward direction.
bpsh_cnt	Number of PSH flags set in packets in the backward direction.
furg_cnt	Number of URG flags set in packets in the forward direction.
burg_cnt	Number of URG flags set in packets in the backward direction.
total_fhlen	Total bytes used for headers in the forward direction.
total_bhlen	Total bytes used for headers in the backward direction.

Table 4.1: Description of NetMate statistical features.

[15] https://github.com/DanielArndt/flowtbag

The overall size of the dataset captured and processed through NetMate was 9829 flows with 3569 flows representing VPN traffic and 6260 flows representing Non-VPN traffic. These were labelled *vpn* and *normal*. This overall dataset was split into three separate sets; one for training, one for testing and one for validation of the trained model. The original set was split into 80% training and 20% testing as this was regarded as a generally popular split according to the literature. The resulting testing set was then split further following the same original split, 80/20, resulting in the final testing and validation datasets. The training dataset contained 7863 instances, the final testing dataset contained 1257 instances and the validation dataset contained 253 instances. More detail is provided on the techniques used to accomplish this in section 4.4.2.

4.3 VPN Setup: Streisand on AWS

In the previous chapter, the Azure cloud computing platform was utilised. Specifically, the Azure machine learning studio was used to run machine learning experiments. For this chapter, the cloud platform AWS is used. However, instead of running machine learning experiments, the AWS platform is used to host a virtual machine which acts as a VPN server. The server that was chosen was the *t2.micro* Elastic Compute Cloud (EC2) instance which is one of the more basic types, but more than adequate to run a fully featured VPN server. It contains one virtual CPU core and one gigabyte of RAM. The software used to setup the server to allow it to provide VPN functionality is called Streisand[16]. Streisand sets up a new remote server with the Ubuntu 16.04 operating system that can run various services such as L2TP/IPsec, OpenVPN and other methods of tunnelling network traffic via VPN. The setup is heavily automated, relying on an automation tool named Ansible that is typically used to provision and configure files and packages on remote servers. The only input required from the user is to choose a cloud provider, physical region for the server and the API information for the cloud platform that the user wishes to set the server up on.

Once this information has been provided, the script begins the creation and initial setup process for the remote server, installing the required software and tools needed. Once

[16] https://github.com/StreisandEffect/streisand

the server has been fully set up, a number of local files are created on the user's local computer which contain instructions on getting started. There is also an option to forgo the cloud providers and perform the setup on a local computer, however this was not investigated. For the experiments run as part of this chapter, the OpenVPN protocol is the type of VPN being tested. The reason for choosing OpenVPN is because it is quite popular in part due to its simple configuration. Servers and clients are widely supported across many different platforms i.e. an OpenVPN server running on a Linux/Unix based host can be accessed by a Windows client and vice versa. There are also client applications available for mobile platforms such as Android and iOS, making it suitable for on-the-go VPN use.

4.4 Weka Experiment

For the development of a VPN classifier, it was decided that a change of platform was required. During the early testing mentioned in section 4.2 it was found that Azure Machine Learning studio tended to overfit during training even with the updated TCP flow statistics dataset. This issue was likely a result of the continuous development of the Azure platform and until this unique issue could be resolved, the Azure platform was not able to handle the algorithm being researched in the context of this experiment.

The Waikato Environment for Knowledge Analysis (Weka) workbench was the option chosen based on its reputation as a very powerful tool for understanding and analysing machine learning algorithms. Even though in chapter 3, cloud computing was found to be a preferable choice of platform for the experiments conducted in this research, because Azure was found to be lacking in this situation, no other cloud platform was capable of dealing with the task at hand. Given this it was determined to go with a local platform because although these are often limited to the hardware doing the computations, it meant that during the experiment, access to robust machine learning algorithms was ensured.

The Weka workbench is a collection of standard machine learning algorithms and data pre-processing tools. It is designed to allow researchers to quickly apply existing methods of machine learning to new datasets (Frank et al., 2016). Weka includes methods for many types of machine learning problem including: regression, classification, clustering and attribute selection. Weka includes a number of ways of

setting up experiments: *Explorer, Knowledge Flow, Experimenter and Workbench*. *Explorer* is a graphical user interface which provides access to all of the facilities of Weka using menu selection and forms. *Knowledge flow* is an interface that allows you to visualise and control the stream of data when using larger datasets. A drawback of *Explorer* is that datasets are loaded in their entirety to the computers RAM, meaning that datasets that need a larger amount of memory than the computer can provide will not be able to be used. *Knowledge flow* enables a researcher to specify a data stream by connecting components representing data sources, pre-processing tools, learning algorithms, evaluation methods and visualisation modules. If the filters and learning algorithms are capable of incremental learning, then the dataset will be loaded into memory in increments causing memory to be saved. *Experimenter* is designed to answer the question of which learning algorithms and parameters values work best for the given problem. This can be accomplished manually using *Explorer.* However, *Experimenter* allows the researcher to automate the process by making it easy to run different classifiers with different parameters on multiple datasets, collect the results and performance statistics and then analyse them to see what combination works best for the given problem. The last interface is called *Workbench.* It is a unified graphical interface which incorporates features from the other three into one application. It is highly configurable and allows for the creation of a highly tailored interface.

The method used for experiments described in this chapter is based on *Explorer.* This is the most straight-forward interface to using Weka and can be used to load in datasets, run experiments and analyse the results. Weka's native data storage method is the ARFF format, however it provides methods to convert data to ARFF from spreadsheets and databases. It can also accept comma-separated value (CSV) files, ASCII MATLAB files, LIBSVM etc. and also provides methods to convert them to the ARFF format. CSV files that have had the ARFF attribute information added to them manually can also be accepted as long as Weka is able to interpret the ARFF headers correctly. The ARFF header helps Weka to identify what is an attribute, what is an instance of the data and what are the classes (if any).

Once the data has been loaded into Weka, it can be modified using the *Pre-process* tab. Modifications include the ability to remove attributes, add instances manually and apply various filters to the entire dataset. Modified versions of the dataset can be saved

to their own file, leaving the original dataset intact for future use or modification. One of the most useful modifications is the ability to split the original dataset into separate training, testing and validation sets using the provided filters which is discussed in section 4.4.2. The next tab is *Classify* and it is here that the various machine learning algorithms are trained to perform classification or regression and evaluate the results. The dataset that is loaded into the *pre-process* tab is seamlessly transferred across to the *classify* tab. If not already done, the dataset can be split into training and testing here using the "Test options" selections or, if already completed, the test set can be specified.

The type of classification or regression algorithm can be chosen here as well as from the large selection that Weka provides, including Linear Regression, Multilayer Perceptron, Naïve Bayes and C4.5 decision tree algorithms. All of the classifiers are adjustable via another Weka dialogue which enables customisation via drop down menus and textboxes. Once setup the algorithm can be trained and tested by pressing the start button and once the model has been successfully trained and tested, the results are shown in the "Classifier output". The *Cluster* and *Associate* tabs were not used during the experiments described in this chapter. The *Select attributes* tab gives access to several methods for attribute selection. This involves an attribute evaluator and a searching method. Both are selected and configured in the same way that options are chosen and configured in the other tabs. Selection can be performed using either the full dataset or by using cross-validation. The full dataset option was used for this chapter's experiments. This allows the researcher to perform feature selection using a number of different feature selection algorithms. The feature selection performed on the dataset used in this chapter is discussed further in section 4.4.1.

4.4.1 Feature Selection

The feature selection model used was the Weka model *CorrelationAttributeEval* which is a model that is based on Pearson's Correlation Coefficient model. This is a measure of the linear correlation between two variables. The output of the model is a value between +1 and -1, where +1 is total positive linear correlation, 0 is no linear correlation and -1 is total negative linear correlation. In Weka, the search method

Ranker is required to run *CorrelationAttributeEval*. *Ranker* ranks attributes by their individual evaluations i.e. from highest positive linear correlation to lowest negative linear correlation. It provides options to set a threshold by which attributes can be discarded, with the default being that no attributes are discarded. Through trial and error, the threshold for the experiments run in this chapter was set to 0.5. This threshold is the cut-off point for whether an attribute of the data is kept as a feature or discarded.

Attribute Name	Correlation Coefficient
total_fpackets	0.561
total_fvolume	0.544
max_fpktl	0.644
max_bpktl	0.724
duration	0.742
mean_active	0.677
max_active	0.57
std_active	0.55
fpsh_cnt	0.587
total_fhlen	0.561

Table 4.2: Correlation Coefficients for selected attributes

The result of this selection was a reduction from 44 features to the 10 features that were calculated to have a linear correlation above 5, a list of which is shown in table 4.2. The lowest correlation was 0.544 for *total_fvolume* and the highest was 0.742 for *duration.*

4.4.2 Resampling the dataset into training, testing & validation sets

The *resample* filter can be found under the “instance” folder under the “supervised” folder. The original, full dataset was edited to create a training dataset of 7863 by resampling using the options outlined in figure 4.6. The 80/20 percent split can be seen in *sampleSizePercent* as “80”.

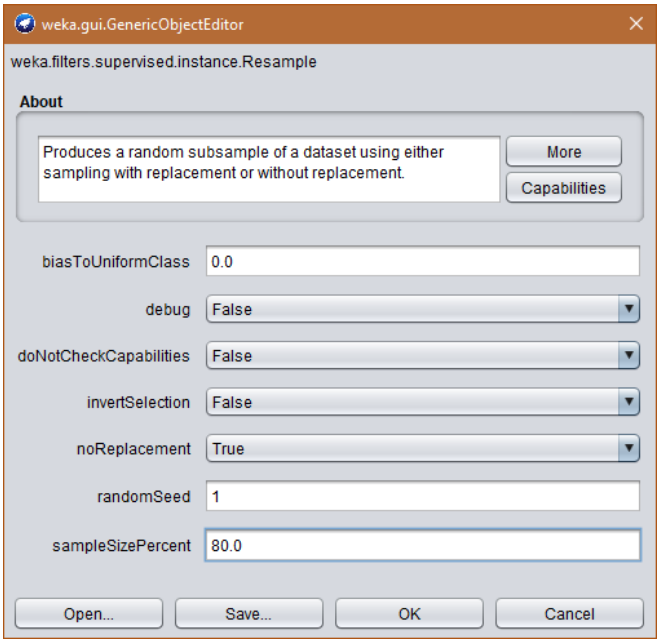

Figure 4.6: The Weka Resample dialogue

To create the testing dataset, the original dataset was again resampled using the same criteria, but the *invertSelection* option was changed from false to true. This gives the opposite output of the first run and the output is a dataset of 1510 instances which is the 20% from the 80/20 split. These instances were then again resampled following the above steps in order to create the final testing dataset and the final validation dataset. Again, this followed the 80/20 percent split, with the 80% split being the testing set and the remaining the 20% being the validation set. The resulting datasets were 1257 instances for the testing dataset, or approximately 12% of the original dataset, and 253 instances for the validation dataset, approximately 2-3% of the original dataset.

4.4.3 Neural Network Setup

The Weka model used for classifying whether the instances from the dataset are traffic coming from a VPN or not is the *MultilayerPerceptron* model. This model is based on a standard artificial Neural Network that is trained using back propagation. This model offers a large amount of customisation, with options to build a network by hand, let an algorithm build the network or a mixture of both. Figure 4.7 shows the

configuration used to setup the neural network model used for the classification experiments.

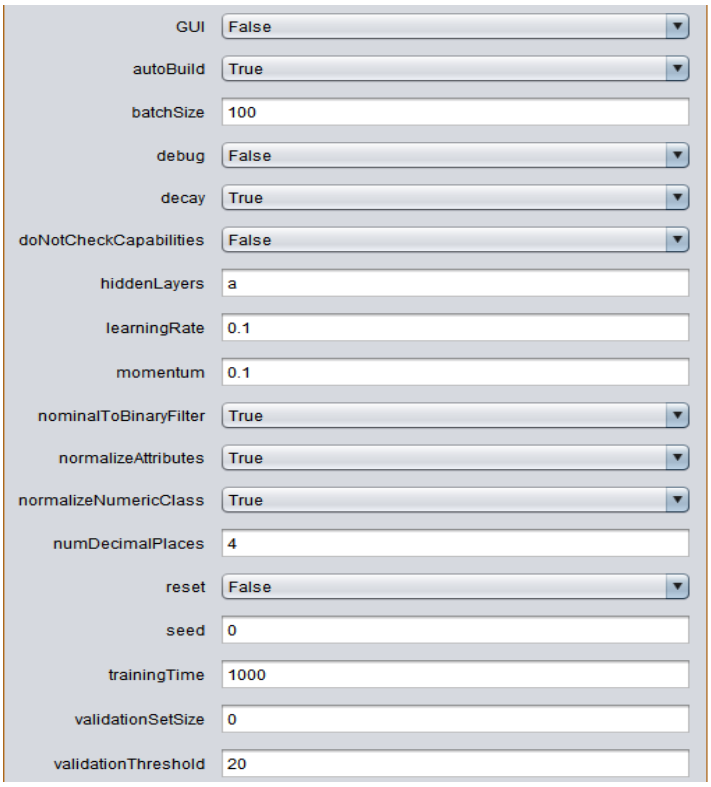

Figure 4.7: Neural Network Weka Configuration

This setup was found to be the best performing configuration when compared to other networks with different configurations with regards to accuracy, training time and avoiding the problem of overfitting the data. Ideally for this model to be used in a real-world application, the training time needs to be kept to a minimum whilst preserving as much accuracy as possible. For the purposes of classifying VPN and non-VPN traffic it was decided to allow Weka to create a fully connected network in order to leverage all of the data instead of having parts of the data be degraded from the network over time. Semi-connected networks are capable of answering questions in a more creative or chaotic way, but tend to lose focus on the overall problem (Theiler, 2014). For a binary classification problem, it is often better to pursue simpler models as less ambiguity is injected into the networks learning phase.

This is accomplished by using the two options *autoBuild* and *hiddenLayers*. *Autobuild* is the option which instructs Weka whether to build a fully connected network or not

with the two options being true or false. *HiddenLayers* is where the hidden nodes of the network are defined.

The value shown in figure 4.6 is one of the provided wildcard values. ‘a’ creates a hidden layer by summing together the number of attributes and classes and then dividing the total in half. So, for 10 attributes and two classes, the number of hidden layers is set to six. Figure 4.8 shows the completed network, ready to be trained using the training dataset.

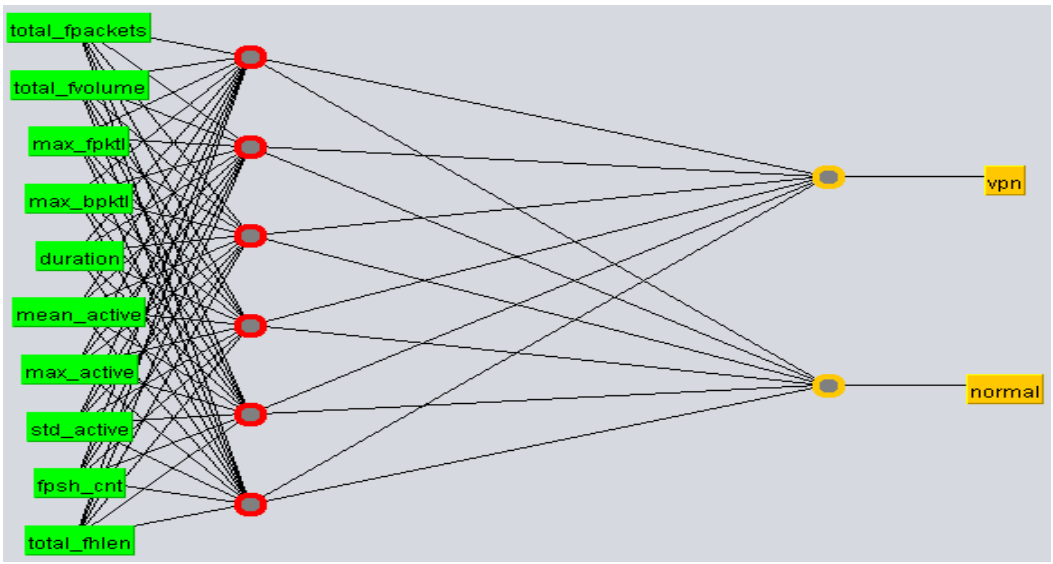

Figure 4.8: Fully connected Neural Network

Once the model is configured, it is ready to train using the dataset currently loaded into the “Preprocess” tab. This network was trained using the above options and the total time taken to train the network and build the classification model was approximately 10 seconds using an 8-core processor. Testing was completed a few seconds later using the testing dataset. Validation of the result using the validation dataset was completed by loading it in as a test set and then re-evaluating the already trained model.

4.4.4 Results

Tables 4.3 and 4.5 show the results as measured by Weka for the validation tests and for the final blind test respectively. Tables 4.4 and 4.6 display the confusion matrices for the two sets.

Correctly Classified Instances	1178 / 1257 (93.7152%)
Incorrectly Classified Instances	79 / 1257 (6.2848%)
Average True Positive Rate	0.937
Average False Positive Rate	0.081
Average Precision	0.937
Average Recall	0.937
Average F-Measure	0.937

Table 4.3: Validation test results

The results shown in table 4.3 shows that the overall accuracy of detection for the neural network in the post-training test was approximately 94%. That is 1178 correctly classified instances out of a testing set of 1257.

Classified as	*VPN*	*Normal*
VPN	408	48
Normal	31	770

Table 4.4: Confusion Matrix for Validation test

Table 4.4 shows the confusion matrix for the validation test. It provides details on the correctly and incorrectly classified instances and how they are distributed as true positive and negative and false positive and negative. The goal is to keep the false positive and false negative as low as possible and table 4.4 shows that this has indeed been accomplished. The number of false positives (i.e. *Normal* traffic that has been incorrectly classified as a *VPN*) was 31 instances. The number of false negatives (i.e. *VPN* traffic that has been incorrectly classified as *Normal*) was 48 instances.

Correctly Classified Instances	232 / 253 (91.6996%)
Incorrectly Classified Instances	21 / 253 (8.3004%)
Average True Positive Rate	0.917
Average False Positive Rate	0.113
Average Precision	0.917
Average Recall	0.917
Average F-Measure	0.916

Table 4.5: Testing results

Table 4.5 shows the results for the final blind test results. The training test was performed on data that had been kept separate from the training process. The data was classified by the trained model as new data that it had never encountered before, therefore imitating real world conditions. The result was an accuracy rating of approximately 92% or 232 correctly classified instances out of 253.

Classified as	*VPN*	*Normal*
VPN	78	14
Normal	7	154

Table 4.6: Confusion Matrix for Testing results

Table 4.6 shows the confusion matrix for the training test. As explained for table 4.4, this provides details on the correctly and incorrectly classified instances and how they are distributed as true positive and negative and false positive and negative. The number of false positives for the validation test was seven and the number of false negatives was 14.

In the context of this model operating under real-world conditions on a live network, the validation results show that for every 100 network packets classified by the model, approximately 94 of them will be classified correctly and 6 will be classified incorrectly. For the separate training test results, there would be approximately 8

errors. As with the proxy detection model discussed in chapter 3, the model is shown to be slightly too lenient in classifying positive instances, with more false negatives than there are false positives.

4.5 OpenVPN using Stunnel

Stunnel[17] is an open source, multiplatform application that is designed to add SSL/TLS encryption capability to clients and servers that do not natively support the SSL/TLS protocols. While OpenVPN itself has support for SSL/TLS, techniques such as Deep Packet Inspection (DPI) have the potential to detect OpenVPN when using SSL/TLs (Kazemi & Fanian, 2015; Deri et al., 2014). Stunnel can be used to overcome this and present the traffic to DPI frameworks as normal SSL web traffic running on port 443. This gave rise to the question of whether a similar method of classification that was used to classify OpenVPN traffic using a neural network could also be trained to recognize OpenVPN traffic that was using Stunnel. To use Stunnel, the user must install and configure the application on both the OpenVPN server and on whatever OpenVPN client they are using to connect to the VPN. On Linux this involves installing the application by downloading the *stunnel4* package, creating and sharing a new OpenSSL certificate between the client and the server, creating and editing Stunnel config files and configuring the firewalls of both the server and client to allow the Stunnel traffic to be transported.

4.5.1 Dataset

As with the previous experiments, a dataset containing network traffic from Stunnel OpenVPN connections and non-VPN traffic is required to train the neural network. With the ground work already done with the setup of the OpenVPN server on AWS for the previous experiment, this was relatively simple. The Streisand VPN package also contained everything necessary to setup Stunnel for use with OpenVPN, only requiring a few configuration files to be modified. Once the VPN was setup and the connection stable, capture of the network traffic began using the same method as used for the OpenVPN data capture. Wireshark was used to capture network packets; the

[17] https://www.stunnel.org/

VPN was set to disconnect and reconnect every 10 minutes and automatic browsing script was used to generate traffic from the same selection of websites. Once the packets were captured, they were processed using the TCP flow export tool NetMate in order to gain flow statistics of the new data. The result of this data capture was a total dataset of 3,952 samples, of which 1,931 were Stunnel OpenVPN and 2,021 were non-VPN. This dataset was then loaded into Weka.

4.5.2 Feature Selection

Feature selection was applied to the capture data in order to reduce the number of features produced by NetMate. Again, the same Weka technique used for the OpenVPN experiment was used. This was the *CorrelationAttributeEval* model which was also operating under the same threshold of 0.5. The resulting features are displayed in Table 4.7.

Attribute Name	Correlation Coefficient
min_fpktl	0.992
duration	0.937
max_fpktl	0.913
max_idle	0.78
max_biat	0.763
std_idle	0.719
max_fiat	0.673
mean_idle	0.575
min_idle	0.562
mean_fpktl	0.561
mean_active	0.512
max_active	0.511
std_fpktl	0.506

Table 4.7: Correlation Coefficients for selected Stunnel attributes

The feature selection for the Stunnel data appears to be largely different to the features selected for the original VPN dataset. Some attributes make a reappearance, such as *duration*, but with a different correlation coefficient. Some of the attributes selected this time haven't been seen before which would seem to indicate that there is a difference in how Stunnel modifies the OpenVPN connection. Following the same steps used in the previous experiment, the dataset was resampled into separate training, testing and validation sets. The training set contains 3160 samples, the testing set contains 633 samples and the validation set contains 127 samples after resampling.

4.5.3 Neural Network setup

For this experiment the goal was to examine how well the model developed in the previous experiment could also perform the same with network traffic from a different source. Therefore, the neural network model used in the previous experiment was reused without any modification. Figure 4.6 shows the configuration of this model. Weka was instructed to create a fully connected network with the hidden layer being defined by a. 'a' is a provided wildcard which creates a hidden layer by summing together the number of attributes with the number of classes and divide the result by 2. In this instance there are 13 attributes and 2 classes which results in 15 divided by 2 which is 7.5. Weka rounds down to the nearest whole number so the number of hidden nodes is set to 7. Figure 4.9 shows this completed network.

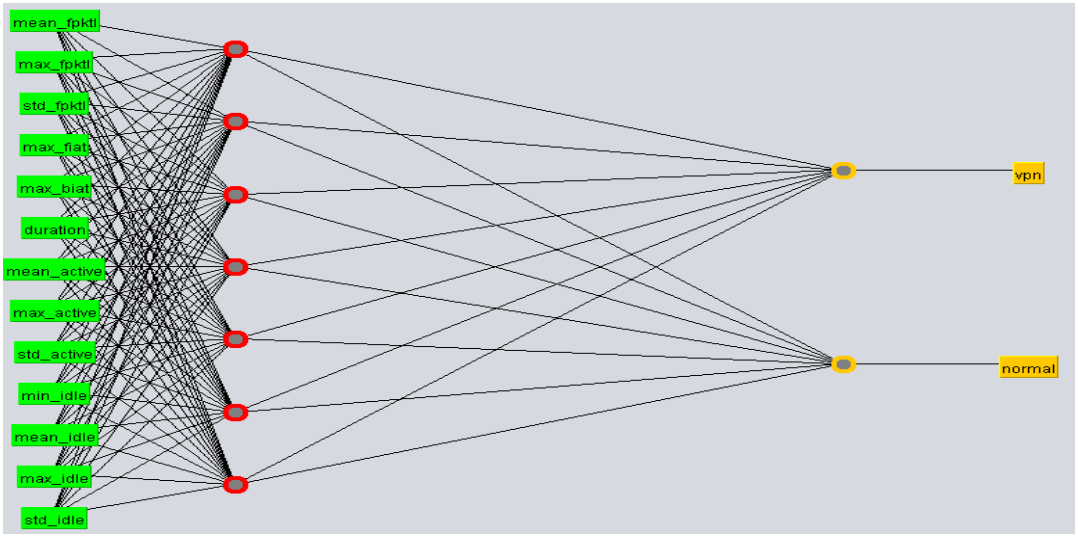

Figure 4.9: Fully connected Neural Network for Stunnel experiment.

Once at this stage the model is ready to be trained using the dataset. In the previous experiment, the model was trained, tested and validated using three resampled sets of data. The same method was used for this model with additional tests being run using 10-fold cross-validation and Leave One Out Cross Validation (LOOCV). On initial testing using these validation methods, the results gathered showed that the model was getting unrealistically high accuracy, possibly showing signs of overfitting of the model to the problem. To remedy this, the learning rate and then the momentum of the model were lowered from 0.1 to 0.01.

4.5.4 Results

Tables 4.8, 4.9 and 4.10 show the results of each validation method used once the neural network had been finally trained using the updated configuration. Tables 4.11, 4.12 and 4.13 show the confusion matrices for each of the tests. Figure 4.10 shows a bar chart comparing the overall accuracies of each test to a test run without any rules applied.

The ZeroRules method in Weka displays what the results would be in the event where everything is classified as one of the classes, in this case that was the normal class. Compared to the zero rules result, the neural network performs very well.

Correctly Classified Instances	125 / 127 (98.4252%)
Incorrectly Classified Instances	2 / 127 (1.5748%)
Average True Positive Rate	0.968
Average False Positive Rate	0.000
Average Precision	1.000
Average Recall	0.968
Average F-Measure	0.984

Table 4.8: 80/20 split Validation test results

Table 4.8 shows the results gathered from Weka for the test that used an 80/20 percentage split on the dataset to create separate training, testing and validation sets. The results shown are taken from the final validation set test, which uses data that was kept separate from the training and tuning of the model in order to simulate as close as possible the real-world performance of the model. The overall accuracy of the model was shown to be 98.42%.

Correctly Classified Instances	3869 / 3952 (97.8998%)
Incorrectly Classified Instances	83 / 3952 (2.1002%)
Average True Positive Rate	0.969
Average False Positive Rate	0.012
Average Precision	0.987
Average Recall	0.969
Average F-Measure	0.978

Table 4.9: 10 fold Cross Validation test results

Table 4.9 shows the results gathered from the test that used 10-fold cross validation to validate the model. For validation of this model the dataset was split into 10 equally sized subsamples or folds. Of these 10 subsamples, one is retained as the validation data for testing of the model and the remaining 9 subsamples are used as training data. This process is then repeated 10 times so that each of the folds is exactly once as the validation data. These results are then averaged to provide a single estimation of the performance of the model. The overall accuracy as shown by this validation is shown to be 97.89%.

Correctly Classified Instances	3866 / 3958 (97.8239%)
Incorrectly Classified Instances	86 / 3952 (2.1761%)
Average True Positive Rate	0.968
Average False Positive Rate	0.012
Average Precision	0.987
Average Recall	0.968
Average F-Measure	0.978

Table 4.10: Leave One Out CrossValidation test results

Table 4.10 shows the results gathered from the test that used Leave One Out cross validation to validate the model. LOOCV involves a similar process to 10-fold Cross Validation where, instead of splitting the data into equal sized folds, only one sample is retained as the validation data, with the rest being used as training data. This process is repeated as many times as there are samples in the dataset i.e. until every single sample has been used as the validation data once. The overall accuracy achieved using this validation method was found to be 97.82%.

Classified as	*VPN*	*Normal*
VPN	60	2
Normal	0	65

Table 4.11: Confusion Matrix for 80/20 split Validation test

Table 4.11 shows the confusion matrix for the test that used an 80/20 percentage split on the dataset. It shows 60 samples were correctly identified as VPN, 65 samples were correctly identified as non-VPN and 2 were incorrectly identified as non-VPN. Interesting is the lack of samples that were incorrectly identified as VPN.

Classified as	*VPN*	*Normal*
VPN	1872	59
Normal	24	1997

Table 4.12: Confusion Matrix for 10 fold Cross Validation test

Table 4.12 shows the confusion matrix for the test that used 10-fold cross validation. It shows 1872 samples were correctly identified as VPN, 1997 samples were correctly identified as non-VPN, 24 samples were incorrectly identified as VPN and 59 samples were incorrectly identified as non-VPN.

Classified as	*VPN*	*Normal*
VPN	1870	61
Normal	25	1996

Table 4.13: Confusion Matrix for Leave One Out Cross Validation test

Table 4.13 shows the confusion matrix for the test that used LOOCV for validating the model. It shows 1870 samples were correctly identified as VPN, 1996 samples were correctly identified as non-VPN, 25 samples were incorrectly identified as VPN and 61 samples were incorrectly identified as non-VPN.

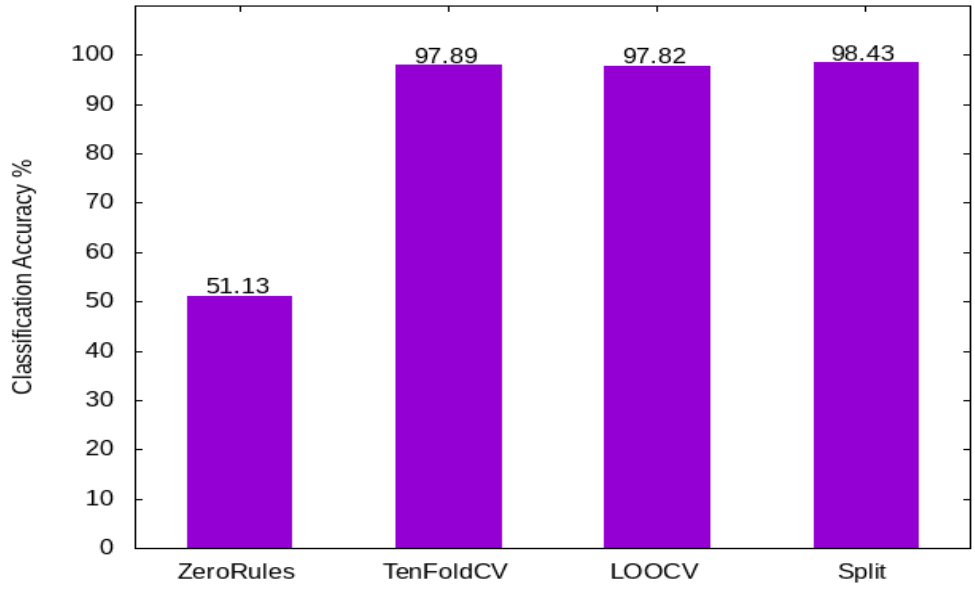

Figure 4.10: Graph comparing accuracies of different validation techniques against ZeroRules

As shown in table 4.8, the 80/20 split validation method was able to achieve an accuracy rate of 98.43%. Initially this would suggest that the 80/20 training and test split provides the best model, because the overall number of samples in the validation set is comparatively low, the results may not be reliable. This leaves the two types of cross validation to be compared to each other. 10-fold cross validation is one of the more popular forms of cross validation and is widely used. LOOCV is essentially cross validation where the number of folds that the data is sub-divided into is the same as the total number of samples in the dataset, in this case that would be 3952 folds. In the results the overall accuracies of the two methods are very close to one another. However, LOOCV has a much higher computation time when compared to 10-fold cross validation despite the individual fold computation time being lower. When 10-fold validation is used the model only has to be trained and tested once for each of the 10 folds, the model in this case must be trained and tested 3952 times when using LOOCV. Because the results of the two validation techniques are so close to one another, this means the benefits of LOOCV are possibly worthless.

So, if we take the result of the 10-fold cross validation of 97.89% as the best indicator, it can be said that the neural network has the ability to accurately distinguish between an OpenVPN connection making use of Stunnel and normal non-VPN traffic.

However, as noticed with the previous OpenVPN experiment, the confusion matrices for all of the validation methods used this time round show that the model is slightly too lenient, with a higher number of false negatives than false positives.

4.6 Validation testing

The work that will form the basis of this comparison will be that of (Draper-Gil et al., 2016) previously mentioned in the introduction to this chapter. This work introduces a classification method for classifying encrypted and VPN traffic from various sources, the main focus being Voice over Internet Protocol (VoIP). Additionally, the paper also evaluates the performance of the approach on other types of Internet traffic such as browser-based, mail-based and peer-to-peer traffic. The paper uses two different machine learning algorithms, C4.5 and *k*-nearest neighbour (KNN).

The results of the browser-based approach presented in (Draper-Gil et al., 2016) are the only results used for validation of the models presented in Section 4.4.3 and 4.5.3 as browser-based traffic is the primary scope of the research conducted using the developed models. In the experiments carried out in (Draper-Gil et al., 2016), two scenarios is examined.

In the first scenario where the goal is to classify encrypted traffic as well as identifying whether the source was a VPN by first distinguishing between VPN and Non-VPN traffic following by classifying the traffic into one of several sub-types. To accomplish this, the dataset used was split into two separate datasets with one containing Non-VPN traffic and the other containing VPN traffic. In the second scenario a dataset combining the two sets of data from the first scenario are used, with the VPN identification and classification of traffic being accomplished at the same time.

The results for the first scenario highlight the precision and recall of the classification of the different types of traffic. For browser-based traffic, the best results obtained were from the C4.5 algorithm. The precision obtained was approximately 0.88 and 0.93 for the VPN and Non-VPN classifiers respectively giving a mean average

precision of 0.905 that was calculated from the results presented in (Draper-Gil et al., 2016).

The results for the second scenario highlight the precision and recall of the classification of whether the traffic in question originated from a VPN and what type of traffic it is. For browser-based traffic, the best results attained were again obtained from the C4.5 algorithm with the precision being approximately 0.81 and 0.82 for the VPN and Non-VPN classification respectively giving a mean average precision of 0.815 that was again calculated from the results presented in (Draper-Gil et al., 2016). These results show that the approach can produce a classifier to characterise encrypted non-VPN and VPN traffic.

The goals of the experiments conducted in Chapter 4 of this thesis were the creation of machine learning models based on neural networks that were capable of characterising VPN and non-VPN traffic. The first model that was developed focused on classifying normal and standard OpenVPN traffic. The second model that was developed focused on classifying normal and Stunnel OpenVPN traffic. The results obtained for the first model show an average precision of 0.917 when classifying traffic as either VPN or non-VPN. The results obtained for the second model show an average precision of 0.987 when classifying traffic as either Stunnel VPN or non-VPN.

Comparing the results obtained from (Draper-Gil et al., 2016) and the ones obtained from the two proposed models, it is clear that the proposed models perform better when applied to the selected scenario.

4.7 Summary

This chapter explored the development of a neural network model that was capable of classifying network traffic as VPN traffic or not by using TCP flow statistics. Section 4.2 describes the capture of network packets and the further processing applied to them using NetMate to create flow statistics that can then be compiled into a dataset. Section 4.3 describes the setup of the AWS Virtual Private Server (VPS) used to host the target OpenVPN server.

Section 4.4 follows on from that to describe the experiments carried out using the WEKA machine learning tool. It features an overview of the tool itself and its capabilities, then goes on to describe how it was used to perform feature selection, the setup of the neural network and finally displays and describes the results of the classification.

Section 4.5 then continues the previous work and attempts to apply the same method to a different implementation of OpenVPN which uses the Stunnel application to provide SSL encryption for the VPN. An overview of the dataset captured is given, with details provided on the neural network configuration and how it differs from the original configuration. The results of the classification and different forms of validation and then given.

Section 4.6 presents a validation test using a known state-of-the-art approach as a baseline for the performance of the proposed models.

5. Conclusion and Future Work

5.1 Concluding Summary

The aim of this thesis was to investigate methods that would aid in the detection of anonymising web proxies and VPN technologies that are being used to hide an attacker's identity. While proxies and VPNs have legitimate uses, such as connecting to a business network from a remote location, they are still abused by criminals who use them as a way to commit crimes whilst remaining undetected and unidentified. Without a method to identify when a Proxy or VPN is connecting to a web facing server, businesses could be vulnerable to having their network breached and having data stolen whilst being hindered in their ability to confidently say who stole it. This can be particularly detrimental to websites who deal with customer details and financial records.

There are methods available for inspecting network traffic at the point of ingress and egress. An example of one of these methods is Deep Packet Inspection (DPI). It is closely related to another method called Shallow Packet Inspection (SPI), however SPI only has the ability to inspect the headers of network packets that are used to transport the packets to their destination. DPI goes a step further and inspects those headers and the actual content of the packet, which in the case of a HTTP packet could be a request for data from a website. A counter to DPI is the use of end to end encryption on the content of packets in order to hide those contents from prying eyes. This is done innocently enough with the goal being to stop potential man in the middle attacks from stealing sensitive data such as usernames and passwords or financial details as they are being transmitted. However, proxy and VPN technologies also have the ability to use encryption technologies with the use of IPSec and SSL/TLS. This increases the need for a method to identify these types of network traffic. Machine learning techniques are one way in which to accomplish this.

The chapters detail the steps taken to develop a machine learning technique for identifying proxy and VPN network traffic. These chapters are summarised below.

Chapter 2 provides a background of the technologies investigated as well as the state of the art in the use of them and detecting their use. The chapter first gives a background on the various types of Proxy that are available and explaining how Proxy have both legitimate and illegitimate uses. A number of different anonymising Proxy technologies were described with some details given on how they are developed. The second section of the chapter is focused on giving a background of different VPN technologies and how they've been developed and improved. Next a background on Intrusion Detection Systems is provided with a review on how machine learning techniques have been integrated with them to improve their detection rates. Finally, a background is provided for Neural Networks to aid in the understanding of how they operate.

Chapter 3 created the neural network model that was capable of classifying Anonymising Proxy network traffic versus normal traffic not being routed through an Anonymising Proxy. The chapter begins by outlining the hardware setup used to generate network traffic and the reasons for the choices made. Then the chapter begins describing the steps taken to generate the dataset needed to train and test the neural network model. This involved using the Proxy client described in the hardware setup section to generate and capture network traffic from anonymising web traffic sources and network traffic from non-Proxy sources. The capture used automated browsing and capture scripts that were written in Python. The features of the dataset resulting from the network traffic capture took the form of TCP header details. This dataset was used alongside the Microsoft Azure Machine Learning Studio to train, tune and test a binary class Multi-layered Perceptron Neural Network.

As tuning of the model progressed, it was found that some features were causing the model to overfit to the data and these were then removed. Once overfitting was reduced a completed model could be trained and was tested on data from the dataset that had purposely been kept separate from the training and tuning processes to reduce any bias that may have formed in the model. The results of the validation test showed that the model was capable of classifying network traffic as either Anonymising Proxy traffic or as non-Proxy traffic and the concluding results section details the results obtained.

Chapter 4 involves a follow-up investigation of whether similar techniques to those used in Chapter 3 could be used to identify and classify network traffic belonging to a VPN. The chapter begins by outlining how VPNs can be problematic when used as an identity hiding tool and gives justification for the investigation. Discussed next is the need for an additional dataset which is needed to train the Neural Network model. The tools used to capture it were a Virtual Machine running Ubuntu 18.04, Wireshark and a variation of the automated browsing script from chapter 3. This dataset was initially based off the same theory as the one underlying the Proxy dataset, that the TCP header details would provide enough of a pattern to allow for detection of VPN network traffic.

However, the Neural Network model was unable to correctly classify the traffic. This led to the decision to investigate the use of TCP flow statistics as features rather than the header details. The flow statistics were calculated by processing the captured network traffic using an application called NetMate. The output of this was a series of features that calculated various time and size related data. The Weka machine toolset was used for all the pre-processing of the flow statistic dataset and for the training and testing of the Neural Network model. The chapter describes in detail the configuration used for the Neural Network experiment and the feature selection process used. The Section 4.4.4 describes the results obtained from this series of experiments.

Building off this initial VPN experiment, Section 4.5 explores the problem further by using a different variation of OpenVPN which uses an application called Stunnel to provide an encrypted connection. Another dataset was captured containing new traffic data which was processed through NetMate to obtain flow statistics. Weka was used again to process the dataset and a Neural Network was trained and tested from the data. The concluding results section describes the results of this further test which show that the slightly modified model was able to classify the Stunnel OpenVPN traffic with a high percentage of accuracy.

5.2 Thesis Contributions

With regards to the objectives laid out in the beginning of this thesis, all have been met in the course of the work undertaken. A full investigation into the detection and classification of Anonymising Proxy and VPN traffic was conducted using a Multi-layered Perceptron Neural Network for the classification of the traffic. From the experiments conducted the Neural Network was found to be able to classify Anonymising Proxy traffic correctly with an overall accuracy of 94.6%. The experiments conducted to classify OpenVPN usage found that the Neural Network was able to correctly identify the VPN traffic with an overall accuracy of 93.71%. The further work done to classify Stunnel OpenVPN usage found that the Neural Network was able to correctly identify VPN traffic with an overall accuracy of 97.82% accuracy when using 10-fold cross validation. This final experiment also provided an observation of 3 different validation techniques and the different accuracy results obtained.

5.2.1 Proxy detection using Neural Network

The first part of this contribution was the generation of a dataset containing traffic from browser sessions using several anonymising proxies and sessions that were not using anonymising proxies. The traffic was labelled as either anonymising proxy traffic or as normal, non-anonymising proxy traffic. The features used were the details of the TCP header contained within each network packet.

The second part of this contribution was the creation of a machine learning model for classification of proxy and non-proxy network traffic trained on the aforementioned dataset. Through the experimental work carried out, it was proven that a neural network was capable of classifying network traffic as either Anonymising Proxy traffic or as non-Proxy traffic. The tests were carried out in such a way that bias was removed where possible when conducting validation tests. Data from the captured network traffic was specifically kept separate from the training and tuning phases of the model creation in order to simulate as close to possible real-world data that the model had not encountered before.

5.2.2 VPN detection using Neural Network

The first part of this contribution was the generation of a dataset containing traffic from browser sessions that were conducted using a VPN connection and sessions that were not conducted using a VPN connection. The traffic was labelled as either VPN traffic or as normal, non-VPN traffic. The features used were statistics gathered from TCP-flows which measured both the number of bytes transferred in both directions as well as the time take to transfer the bytes.

The second part of this contribution was the development of a machine learning model for classification of VPN and non-VPN network traffic trained on the dataset of TCP flow statistics.

Upon successful experiments conducted for the detection of Anonymising Proxy traffic, the focus was extended to include VPN traffic. The VPN technology OpenVPN was chosen as the focus for the experiments, which in turn found that the Neural Network was capable of classifying network traffic as either VPN traffic or as non-VPN traffic.

A further set of experiments which attempted to classify a form of OpenVPN traffic that made use of Stunnel to provide encryption. To facilitate these experiments, a third dataset consisting of TCP flow statistics captured from a combination of normal non-VPN traffic and OpenVPN traffic that was tunneled through Stunnel was created. Using this dataset, the model developed for standard OpenVPN data was trained and tested again on the Stunnel OpenVPN data. Early results showed that this model was overfitting the data, so it was modified to account for this. Once the model was modified, this set of experiments found that a Neural Network trained on the Stunnel OpenVPN data could classify network traffic as either VPN traffic or non-VPN traffic. Again, the experiments were conducted in such as fashion as to eliminate bias where possible. This included keeping a portion of the captured dataset away from the training and tuning phases so it could be used to simulate real world data that the model had never seen before.

5.3 Future Work

This thesis presents a substantial body of work and the research contained within provides novel contributions to the detection of both Anonymising Proxies and VPNs. There are however several directions in which this work could be extended. These are outlined in sections 5.3.1 through to 5.3.5.

5.3.1 Capture of additional data to further test the hypothesis

The datasets captured in the process of the work undertaken for the thesis have been of varying sizes, with the largest being approximately 11000 samples. Time played a large part in how much data was feasible to capture. Going forward, a recommendation could be made to increase the size of the datasets in order to further test the strength of the models created.

5.3.2 Investigation of automatic hyperparameter tuning in Weka

An improvement to the experiments conducted in chapter 4 would be the use of an automatic hyperparameter tuning method similar to that used in the Azure experiments of chapter 3. One such method is provided in the Weka add-on module Auto-WEKA (Thorton et al, 2013). The use of Auto-WEKA has only just been considered because the project has matured to a stable usable point.

5.3.3 Investigation of other machine learning techniques

An advancement of the techniques used in this research would be to investigate the use of Ensemble Learning. Similar to voting systems developed in other fields, Ensemble Learning takes the idea of using the output of multiple base algorithms together into what is known as an ensemble. The benefit of using Ensemble Learning is that more accurate results can be gained as opposed to using each base learning algorithm in isolated fashion. This is due to the diverse nature of ensembles.

One emerging method for forming an ensemble is one known as Stacking. Stacking involves the use of multiple base algorithms which are all trained on the same set of data. The outputs of each base algorithm are aggregated with the actual class and predicted probabilities for an instance. The aggregated outputs are fed into a regression model for each class. The final output of the regression models is combined to form the final classification result (Milliken et al, 2015).

Another possible avenue to explore is the role that Transductive and Matched-Pair machine learning can play individually as well as combined together in this problem as they allow for better metadata to formed from input data (Theiler, 2014).

5.3.3 Obfsproxy with OpenVPN

In order to improve its undetectability against the many forms of censorship, the Tor project set about developing the pluggable transports project (Mazurczyk et al., 2016). One of the results of this project was the pluggable transport Obfsproxy. Obfsproxy is an additional software package that was originally developed to work alongside Tor. Its role was to obfuscate Tor packets by re-encrypting them to conceal the Tor-specific fingerprints that may be present. Today, is can be used as a standalone software that changes traffic signatures to look like traffic that is not normally blocked by methods such as DPI. It can be used in this way to help prevent OpenVPN packets from being detected by DPI. Therefore, some work could be done to investigate whether the methods described in this thesis for detecting OpenVPN could also be applied to detecting the use of OpenVPN alongside Obfsproxy. There is also the possibility of investigating whether it is possible to classify Tor traffic using the same techniques.

5.3.4 Wireguard

Wireguard[18]is a relatively new VPN technology developed to be faster and simpler than IPSec while being more performant than OpenVPN. It is a secure network tunnel that operates at layer three of the network stack implemented as a kernel virtual network interface for Linux (Donenfeld, 2017). There is currently no windows client yet available which limits its popularity, but the author claims to be working on a client. Despite this, Wireguard is gaining traction in industry[19] as well as possible government support in the USA[20]. It may be worth attempting to apply the techniques developed in this thesis to identify Wireguard traffic.

[18] https://www.wireguard.com/
[19]https://arstechnica.com/gadgets/2018/08/wireguard-vpn-review-fast-connections-amaze-but-windows-support-needs-to-happen/
[20] https://www.xda-developers.com/us-senator-pushes-government-use-wireguard-vpn/

5.3.5 Deeper classification with multiple kinds of VPN

The work completed in this thesis focused purely on binary classification of VPN and non-VPN traffic. Another direction that this research could would be investigating the classification of multiple kinds of VPN against non-VPN traffic. This could prove to be a complex task for a standard neural network so it would be worth considering the use of deep learning algorithms such as Convolutional Neural Networks (CNN) or Recurrent Neural Networks (RNN). In recent years deep learning artificial neural networks have won numerous competitions in pattern recognition and machine learning when pitted against more traditional neural networks. (Schmidhuber, 2015). The problem of classifying network traffic boils down to an advanced pattern recognition problem, therefore deep learning artificial neural networks may be well suited to a more complex multiple VPN classification.

Several future areas and directions of work have been identified however there are countless directions that this research could take. The scope of the research could be changed for any network traffic classification problem, provided the dataset is there to train the network. Multi-class classification using deep learning neural network could be a very interesting and useful direction for research to take. However, results obtained over the course of the work undertaken are very positive, with the results of the Stunnel OpenVPN experiment of 97.82% showing that the model is in a good position for evaluation in a real-world VPN detection environment.

References

- Akabogu, C. (2017). Implications Of Mass Media Censorship On The Individual And The Nigerian Society. *International Journal of Communication*. 1(1). 66–74.

- Ali, A. A., Darwish, S. M. & Guirguis, S. K. (2015). An Approach for Improving Performance of a Packet Filtering Firewall Based on Fuzzy Petri Net. *Journal of Advances in Computer Networks.* 3(1). 67–74.

- Anderson, E. L., Steen, E. & Stavropoulos, V. (2017). Internet Use and Problematic Internet Use: A Systematic Review of Longitudinal Research Trends in Adolescence and Emergent Adulthood. *International Journal of Adolescence and Youth.* 22(4). 430–454.

- Arndt, D. (2011). NetMate-Flowcalc [online]. *Daniel Arndt*. [October 4, 2017]. Available from: https://dan.arndt.ca/projects/netmate-flowcalc/

- Attneave, F. & B., M. (1950). The Organization of Behavior; A Neuropsychological Theory. *The American Journal of Psychology*. 63(4). 633–642.

- Bengio, Y. & others. (2009). Learning Deep Architectures for AI. *Foundations and trends® in Machine Learning*, 2(1), 1–127.

- Berger, T. (2006). Analysis of Current VPN Technologies, in: *First International Conference on Availability, Reliability and Security (ARES'06)* [online]. Vienna, Austria: IEEE. Available from: DOI: 10.1109/ARES.2006.30

- Bihis, M. & Roychowdhury, S. (2015). A Generalized Flow for Multi-Class and Binary Classification Tasks: An Azure ML Approach, in: *2015 IEEE International Conference on Big Data (Big Data)* [online]. Santa Clara, USA: IEEE. Available from: DOI: 10.1109/BigData.2015.7363944

- Bujlow, T., Riaz, T. & Pedersen, J. M. (2012). A Method for Classification of Network Traffic Based on C5.0 Machine Learning Algorithm, in: *2012 International Conference on Computing, Networking and Communications, ICNC'12* [online]. IEEE. [March 5, 2018]. Available from: http://ieeexplore.ieee.org/document/6167418/

- Cardellini, V., Yu, P. S. & Huang, Y.-W. (2000). Collaborative Proxy System for Distributed Web Content Transcoding, in: *Proceedings of the ninth international conference on Information and knowledge management - CIKM '00* [online]. New York, New York, USA: ACM Press. [December 13, 2017], Available from: http://portal.acm.org/citation.cfm?doid=354756.354861

- Chang, C. Y. & Chen, M. S. (2003). On Exploring Aggregate Effect for Efficient Cache Replacement in Transcoding Proxies. *IEEE Transactions on Parallel and Distributed Systems* [online]. 14(6). 611–624. [December 13, 2017]. Available from: http://ieeexplore.ieee.org/document/1206507/

- Chaum, D. L. (1981). Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM*, 24(2), 84–90. [March 3, 2018]. Available from: http://portal.acm.org/citation.cfm?doid=358549.358563

- Cisco. (2006). Access Control Lists: Overview and Guidelines [online]. *Cisco.* Available from: http://www.cisco.com/c/en/us/td/docs/ios/12_2/security/configuration/guide/fsecur_c/scfacls.html

- Cisco. (2017). The Zettabyte Era: Trends and Analysis [online]. *Cisco.* [May 2, 2018]. Available from: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html#_Toc484556819

- Cisco. (2018). Encrypted Traffic Analytics [online]. *Cisco*. [January 12, 2018]. Available from: https://www.cisco.com/c/dam/en/us/solutions/collateral/enterprise-networks/enterprise-network-security/nb-09-encrytd-traf-anlytcs-wp-cte-en.pdf

- Cobb, J. & ElAarag, H. (2008). Web Proxy Cache Replacement Scheme Based on Back-Propagation Neural Network. *Journal of Systems and Software* [online]. 81(9). 1539–1558. [March 30, 2017]. Available from: http://www.sciencedirect.com/science/article/pii/S016412120700249X

- Crist, E. F. & Keijser, J. J. (2015). *Mastering OpenVPN*. Packt Publishing Ltd.

- Dainotti, A., Pescape, A. & Claffy, K. (2012). Issues and Future Directions in Traffic Classification. *IEEE Network* [online]. 26(1). 35–40. [March 5, 2018]. Available from: http://ieeexplore.ieee.org/document/6135854/

- Deri, L., Martinelli, M., Bujlow, T. & Cardigliano, A. (2014). NDPI: Open-Source High-Speed Deep Packet Inspection, in: *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)* [online]. (pp. 617–622). Nicosia, Cyprus: IEEE. [September 12, 2018]. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6906427

- Dharmapurikar, S., Krishnamurthy, P., Sproull, T. & Lockwood, J. (2003). Deep Packet Inspection Using Parallel Bloom Filters. *IEEE Micro* [online]. 52–61. [March 5, 2018]. Available from: http://ieeexplore.ieee.org/document/1231477/

- Diffie, W. & Hellman, M. (1976). New Directions in Cryptography. *IEEE transactions on Information Theory*. 22(6). 644–654.

- Donenfeld, J. A. (2017). WireGuard: Next Generation Kernel Network Tunnel. in: *24th Annual Network and Distributed System Security Symposium (NDSS 2017), San Diego, USA* [online]. NDSS. Available from: DOI: 10.14722/ndss.2017.23160

- Draper-Gil, G., Lashkari, A. H., Mamun, M. S. I. & A. Ghorbani, A. (2016). Characterization of Encrypted and VPN Traffic Using Time-Related Features, in: *Proceedings of the 2nd International Conference on Information Systems Security and Privacy* [online]. (pp. 407–414). SCITEPRESS - Science and and Technology Publications. [January 17, 2018]. Available from: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0005740704070414

- Durumeric, Z., Ma, Z., Springall, D., Barnes, R., Sullivan, N., Bursztein, E., Bailey, M., Halderman, J. A. & Paxson, V. (2017). The Security Impact of HTTPS Interception, in: *Proc. Network and Distributed Systems Symposium (NDSS)* [online]. (pp. 1–14). San Diego, USA. [June 22, 2017]. Available from: https://jhalderm.com/pub/papers/interception-ndss17.pdf

- Edman, M. & Yener, B. (2009). On Anonymity in an Electronic Society. *ACM Computing Surveys*, 42(1). 1–35. [February 18, 2018]. Available from: http://portal.acm.org/citation.cfm?doid=1592451.1592456

- Farinacci, D., Traina, P., Hanks, S. & Li, T. (1994). Generic Routing Encapsulation over IPv4 Networks [online]. *RFC1702*. 1–4. Retrieved January 17, 2018, Available from: https://tools.ietf.org/html/rfc1702

- Feilner, M. (2006). *OpenVPN : Building and Integrating Virtual Private Networks*. Packt Publishing Ltd.

- Fiaschi, D., Giuliani, E. & Nieri, F. (2017). Overcoming the Liability of Origin by Doing No-Harm: Emerging Country Firms' Social Irresponsibility as They Go Global. *Journal of World Business* [online]. 52(4). 546–563. Available from: https://www.sciencedirect.com/science/article/pii/S1090951616301006

- Finamore, A., Mellia, M., Meo, M. & Rossi, D. (2010). KISS: Stochastic Packet Inspection Classifier for UDP Traffic. *IEEE/ACM Transactions on Networking* [online]. 18(5). 1505–1515. [March 5, 2018]. Available from: http://ieeexplore.ieee.org/document/5443713/

- Fontama, V., Barga, R. & Tok, W. H. (2014). *Predictive Analytics with Microsoft Azure Machine Learning*. Apress.

- Frank, E., Hall, M. A. & Witten, I. H. (2016). *Data Mining: Practical Machine Learning Tools and Techniques*. 4th ed.Morgan Kaufmann.

- García-Teodoro, P., Díaz-Verdejo, J., Maciá-Ferná Ndez, G. & Vá Zquez, E. (2009). Anomaly-Based Network Intrusion Detection: Techniques, Systems and Challenges. *Computers & Security*. 28(1). 18–28.

- Gebhart, G. & Kohno, T. (2017). Internet Censorship in Thailand: User Practices and Potential Threats, in: *Proceedings - 2nd IEEE European Symposium on Security and Privacy, EuroS and P 2017* [online]. (pp. 417–432). IEEE. Available from: http://ieeexplore.ieee.org/document/7961994/

- Geetha, S. & Phamila, A. V. (2016). *Combating Security Breaches and Criminal Activity in the Digital Sphere.* IGI Global.

- Ghosh, A. K., Schwartzbard, A. & Schatz, M. (1999). Learning Program Behavior Profiles for Intrusion Detection. in: *Workshop on Intrusion Detection and Network Monitoring*. 51462, 1-13

- Gourley, D. & Totty, B. (2002). *HTTP: The Definitive Guide*. Beijing. O'Reilly.

- Haddadi, F. & Zincir-Heywood, A. N. (2016). Benchmarking the Effect of Flow Exporters and Protocol Filters on Botnet Traffic Classification. *IEEE Systems Journal* [online]. 10(4), 1390–1401. [May 23, 2017]. Available from: http://ieeexplore.ieee.org/document/6963332/

- Harkins, D. & Carrel, D. (1998). The Internet Key Exchange (IKE) [online]. *RFC2409.* Available from: https://tools.ietf.org/html/rfc2409

- Harmening, J. T. (2013). Virtual Private Networks, in: *Computer and Information Security Handbook.* (pp. 855–867). Elsevier.

- Hawkes-Robinson, W. (2002). SANS Institute - Microsoft PPTP VPN Vulnerabilities - Exploits in Action [online]. Available from: https://www.researchgate.net/publication/235927650_SANS_Institute_-_Microsoft_PPTP_VPN_Vulnerabilities_-_Exploits_in_Action

- Haykin, S. (2004). A Comprehensive Foundation. *Neural Networks* [online]. 2(2004). 41. Available from: http://dl.acm.org/citation.cfm?id=521706

- Hunt, T. (2016). Observations and Thoughts on the LinkedIn Data Breach [online]. *troyhunt.com*. [December 6, 2017]. Available from: https://www.troyhunt.com/observations-and-thoughts-on-the-linkedin-data-breach/

- Jain, V., Appiah, M., Vanniarajan, K. C. & Jain, S. (2011). Secure Tunnel over HTTPS Connection. U.S. Patent 8,086,845.

- Kara, A., Suzuki, T., Takahashi, K. & Yoshikawa, M. (2004). A DoS-Vulnerability Analysis of L2TP-VPN. *The Fourth International Conference onComputer and Information Technology, 2004. CIT '04.* [online]. 397–402. [January 31, 2018]. Available from: http://ieeexplore.ieee.org/document/1357228/

- Kaufman, C., Hoffman, P., Nir, Y., Eronen, P. & Kivinen, T. (2014). Internet Key Exchange Protocol Version 2 (IKEv2) [online] *RFC7296*. Available from: https://tools.ietf.org/html/rfc7296

- Kazemi, K. & Fanian, A. (2015). Tunneling Protocols Identification Using Light Packet Inspection. in: *2015 12th International Iranian Society of Cryptology Conference on Information Security and Cryptology (ISCISC)* [online]. (pp. 110–115). Rasht, Iran: IEEE. Available from: http://ieeexplore.ieee.org/document/7387907/

- Kent, S. & Atkinson, R. (2005). Security Architecture for the Internet Protocol [online] *RFC4301*. Available from: https://tools.ietf.org/html/rfc4301

- Khan, L., Awad, M. & Thuraisingham, B. (2007). A New Intrusion Detection System Using Support Vector Machines and Hierarchical Clustering. *The VLDB Journal* [online]. 16(4). 507–521. [November 2, 2015]. Available from: http://link.springer.com/10.1007/s00778-006-0002-5

- Khriplovich, I. B. & Pomeranskii, A. A. (1998). Equations of Motion of Spinning Relativistic Particle in Electromagnetic and Gravitational Fields. *Journal of Experimental and Theoretical Physics* [online]. 86(5). 839–849. Retrieved from http://arxiv.org/abs/gr-qc/9809069

- King, G., Pan, J. & Roberts, M. E. (2017). How the Chinese Government Fabricates Social Media Posts for Strategic Distraction, Not Engaged Argument. *American Political Science Review* [online]. 111(3). 484–501. [March 1, 2018]. Available from: https://www.cambridge.org/core/product/identifier/S0003055417000144/type/journal_article

- Krithika, R. & Narayanan, J. (2015). Learning to Grade Short Answers Using Machine Learning Techniques, in: *Proceedings of the Third International Symposium on Women in Computing and Informatics - WCI '15* [online]. (pp. 262–271). New York, New York, USA: ACM Press. [February 25, 2016] Available from: http://dl.acm.org/citation.cfm?id=2791405.2791508

- Krizhevsky, A., Sutskever, I. & Hinton, G. E. (2012). Imagenet Classification with Deep Convolutional Neural Networks. in: *Advances in neural information processing systems.* (pp. 1097–1105).

- Kumar, S. T. V., Khanna, S. O. S., Reddy, S. M. P., Reddy, S. G. S. & Reddy, S. G. R. S. (2014). Overview of Emerging Trends in Network Security and Cryptography. *IJEIR* [online]. 3(1). 51–56. Available from: http://www.ijeir.org/index.php/issue?view=publication&task=show&id=250

- Kurose, J. (2014). Information-Centric Networking: The Evolution from Circuits to Packets to Content. *Computer Networks* [online]. 66. 112–120. [January 11, 2018]. Available from: http://www.sciencedirect.com/science/article/pii/S1389128614001455#f0010

- Lawas, J. B. R., Vivero, A. C. & Sharma, A. (2016). Network Performance Evaluation of VPN Protocols (SSTP and IKEv2), in: *2016 Thirteenth International Conference on Wireless and Optical Communications Networks (WOCN)* [online]. (pp. 1–5). IEEE. [February 1, 2018]. Available from: http://ieeexplore.ieee.org/document/7759880/

- Leberknight, C. S., Chiang, M., Poor, H. V. & Wong, F. (2010). A Taxonomy of Internet Censorship and Anti-Censorship. In *Fifth International Conference on Fun with Algorithms* [online]. Available from: http://www.princeton.edu/~chiangm/anticensorship.pdf

- Lee, Y.-D., Leech, M., Ganis, M., Kuris, R., Koblas, D. & Jones, L. (1996). SOCKS: A Protocol for TCP Proxy across Firewalls [online]. *RFC1928.* Available from: https://www.rfc-editor.org/rfc/rfc1928.txt

- Li, B., Erdin, E., Gunes, M. H., Bebis, G. & Shipley, T. (2013). An Overview of Anonymity Technology Usage. *Computer Communications* [online]. 36(12) 1269–1283. [February 18, 2018]. Available from: https://www.sciencedirect.com/science/article/pii/S0140366413001096#b0115

- Ligh, M., Adair, S., Hartstein, B. & Richard, M. (2010). *Malware Analyst's Cookbook and DVD: Tools and Techniques for Fighting Malicious Code*. Wiley Publishing.

- Lin, W.-C., Ke, S.-W. & Tsai, C.-F. (2015). CANN: An Intrusion Detection System Based on Combining Cluster Centers and Nearest Neighbors. *Knowledge-Based Systems* [online]. 78. 13–21. [September 30, 2015] Available from: http://www.sciencedirect.com/science/article/pii/S0950705115000167

- Liu, C., White, R. W. & Dumais, S. (2010). Understanding Web Browsing Behaviors through Weibull Analysis of Dwell Time, in: *Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval - SIGIR '10* [online]. (p. 379). New York, New York, USA: ACM Press. [October 4, 2017], Available from: http://portal.acm.org/citation.cfm?doid=1835449.1835513

- Luotonen, A. & Altis, K. (1994). World Wide Web Proxies, in: *Computer Networks and ISDN Systems, First International Conference on WWW, April*[online]. (pp. 1–8). Geneva, Switzerland. [March 10, 2017] Available from: http://courses.cs.vt.edu/~cs4244/spring.09/documents/Proxies.pdf

- Marshall, J. (2002). CGIProxy [online]. *CGIProxy.* Available from: https://www.jmarshall.com/tools/cgiproxy/

- Mason, A. G. (2004). *CCSP Self-Study: Cisco Secure Virtual Private Networks (CSVPN).* Pearson Higher Education.

- Maughan, D., Schertler, M., Schneider, M. & Turner, J. (1998). *Internet Security Association and Key Management Protocol (ISAKMP)* [online]. [February 1, 2018]. Available from: https://tools.ietf.org/html/rfc2408

- Mazurczyk, W., Wendzel, S., Zander, S., Houmansadr, A. & Szczypiorski, K. (2016). *Information Hiding in Communication Networks : Fundamentals, Mechanisms, Applications, and Countermeasures*. Wiley Publishing.

- McCulloch, W. S. & Pitts, W. (1943). A Logical Calculus of the Ideas Immanent in Nervous Activity. *The bulletin of mathematical biophysics*, 5(4), 115–133.

- Mell, P. and Grance, T., (2011). The NIST definition of cloud computing

- Microsoft. (2012). Microsoft Security Advisory 2743314 | Microsoft Docs. *Microsoft Security Advisory* [online]. *Microsoft.* [January 12, 2018]. Available from: https://docs.microsoft.com/en-us/security-updates/SecurityAdvisories/2012/2743314

- Mikolov, T., Karafiát, M., Burget, L., Černock\`y, J. & Khudanpur, S. (2010). Recurrent Neural Network Based Language Model, in: *Eleventh Annual Conference of the International Speech Communication Association.* (pp. 1045–1048). Makuhari, Chiba, Japan.

- Miller, S., Curran, K. & Lunney, T. (2015a). Securing the Internet through the Detection of Anonymous Proxy Usage. in: *2015 World Congress on Internet Security, WorldCIS 2015*. (pp. 153–158).

- Miller, S., Curran, K. & Lunney, T. (2015b). Traffic Classification for the Detection of Anonymous Web Proxy Routing. *IJISR* [online]. 5(1) 538–545. [March 27, 2018]. Available from: http://infonomics-society.ie/wp-content/uploads/ijisr/published-papers/volume-5-2015/Traffic-Classification-for-the-Detection-of-Anonymous-Web-Proxy-Routing.pdf

- Miller, S., Curran, K. & Lunney, T. (2016). Cloud-Based Machine Learning for the Detection of Anonymous Web Proxies, in: *2016 27th Irish Signals and Systems Conference, ISSC 2016* [online]. (pp. 1–6). IEEE. [February 18, 2018]. Available from: http://ieeexplore.ieee.org/document/7528443/

- Milliken, M., Bi, Y., Galway, L. and Hawe, G., (2015). Ensemble learning utilising feature pairings for intrusion detection, in: *2015 World Congress on Internet Security, WorldCIS* (pp. 24-31). IEEE

- Minsky, M. L. & Papert, S. (1972). *Perceptrons : An Introduction to Computational Geometry*. MIT Press.

- Nguyen, T. T. T. & Armitage, G. (2006). Training on Multiple Sub-Flows to Optimise the Use of Machine Learning Classifiers in Real-World IP Networks, in: *Proceedings - Conference on Local Computer Networks, LCN* [online]. (pp. 369–376). IEEE. [March 5, 2018]. Available from: http://ieeexplore.ieee.org/document/4116573/

- Nguyen, T. T. T. & Armitage, G. (2008). A Survey of Techniques for Internet Traffic Classification Using Machine Learning. *IEEE Communications Surveys & Tutorials* [online]. 10(4). 56–76. [March 5, 2018]. Available from: http://ieeexplore.ieee.org/document/4738466/

- Özyer, T., Alhajj, R. & Barker, K. (2007). Intrusion Detection by Integrating Boosting Genetic Fuzzy Classifier and Data Mining Criteria for Rule Pre-Screening. *Journal of Network and Computer Applications* [online]. 30(1), 99–113. [October 5, 2015]. Available from: http://www.sciencedirect.com/science/article/pii/S1084804505000433

- Pagliery, J. (2014). What Caused Sony Hack: What We Know Now [online]. *CNN*. [December 6, 2017]. Available from: http://money.cnn.com/2014/12/24/technology/security/sony-hack-facts/

- Patel, B., Aboba, B., Dixon, W., Zorn, G. & Booth, S. (2001). Securing L2TP Using IPsec [online]. *RFC3193*. [January 30, 2018]. Available from: https://www.rfc-editor.org/info/rfc3193

- Pathak, A., Patra, B. K., Chakraborty, A. & Agarwal, A. (2015). A City Traffic Dashboard Using Social Network Data, in: *Proceedings of the 2nd IKDD Conference on Data Sciences - CODS-IKDD '15* [online]. (pp. 1–4). New York, New York, USA: ACM Press. [February 25, 2016]. Available from: http://dl.acm.org/citation.cfm?id=2778865.2778873

- Peterson, A. (2014). The Sony Pictures Hack, Explained [online]. *Washington Post*. Available from: https://www.washingtonpost.com/news/the-switch/wp/2014/12/18/the-sony-pictures-hack-explained/

- Pohl, F. & Schotten, H. D. (2017). Secure and Scalable Remote Access Tunnels for the IIoT: An Assessment of OpenVPN and IPsec Performance. in *European Conference on Service-Oriented and Cloud Computing* (pp. 83–90). Springer, Cham.

- Ponnusamy, S. P. & Karthikeyan, E. (2013). Cache Optimization on Hot-Point Proxy Caching Using Weighted-Rank Cache Replacement Policy. *ETRI Journal.* 35(4). 687–696.

- Poushter, J. (2016). Smartphone Ownership and Internet Usage Continues to Climb in Emerging Economies. *Pew Research Center*, 22. 1-44.

- Rawat, V., Tio, R., Nanji, S. & Verma, R. (2001). *Layer Two Tunneling Protocol {(L2TP)} over Frame Relay* [online]. *RFC3070.* [February 13, 2017]. Available from: https://2rfc.net/3070

- Reese, W. (2008). Nginx: The High-Performance Web Server and Reverse Proxy |Linux Journal. *Linux Journal*, 2008(173), 2.

- Rosenblatt, F. (1958). The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, 65(6), 386–408.

- Rowan, T. (2007). VPN Technology: IPSEC vs SSL. *Network Security*, 2007(12), 13–17.

- Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1985). Learning Internal Representations by Error Propagation. *Defense Technical Information Center* [online]. Available from: http://www.dtic.mil/docs/citations/ADA164453

- Russel, S. J. & Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Malaysia: Pearson Education Limited.

- Samuel, A. L. (1959). Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*. 3(3). 210–229.

- Scarfone, K. & Mell, P. (2007). Guide to Intrusion Detection and Prevention Systems (Idps). *NIST special publication*, 800(2007), 94.

- Schmidhuber, J. (2015). Deep Learning in Neural Networks: An Overview. *Neural Networks*, 61, 85–117.

- Schmidt, J. (2012). A Death Blow for PPTP - The H Security: News and Features [online]. *H-Online.* [January 19, 2018]. Available from: http://www.h-online.com/security/features/A-death-blow-for-PPTP-1716768.html

- Schneier, B. (1994). Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish), in: (pp. 191–204). Springer, Berlin, Heidelberg.

- Schneier, B. & Mudge. (1998). Cryptanalysis of Microsoft's Point-to-Point Tunneling Protocol (PPTP). *5th ACM Conference on Computer and Communications Security*, 132–141. ACM.

- Schneier, B., Mudge & Wagner, D. (1999). Cryptanalysis of Microsoft's PPTP Authentication Extensions (MS-CHAPv2), in: *International Exhibition and Congress, Secure Networking*, (pp. 192–203). Duesseldorf: Springer International Publishing.

- Sherry, J., Lan, C., Popa, R. A., Ratnasamy, S., Sherry, J., Lan, C., Popa, R. A. & Ratnasamy, S. (2015). BlindBox. *ACM SIGCOMM Computer Communication Review*. 45(5). 213–226.

- Simpson, W. (1996). PPP CHAP [online]. *RFC1994.* [January 19, 2018]. Available from: https://tools.ietf.org/rfc/rfc1994.txt

- Singh, D. D., Kumar, S. & Kapoor, S. (2011). An Explore View of Web Caching Techniques. *International Journal of Advances in Engineering Sciences*. 1(3). 38–43.

- Soysal, M. & Schmidt, E. G. (2010). Machine Learning Algorithms for Accurate Flow-Based Network Traffic Classification: Evaluation and Comparison. *Performance Evaluation* [online]. 67(6). 451–467. [January 30, 2018]. Available from: https://www.sciencedirect.com/science/article/pii/S0166531610000027

- Stallings, W. & Lawrie, B. (2008). *Computer Security*. Pearson Education.

- Stevanovic, M. & Pedersen, J. M. (2014). An Efficient Flow-Based Botnet Detection Using Supervised Machine Learning, in: *2014 International Conference on Computing, Networking and Communications (ICNC)* [online]. (pp. 797–801). IEEE. [March 15, 2016]. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6785439

- Stibler, S., Brownlee, N. & Ruth, G. (1999). RTFM: New Attributes for Traffic Flow Measurement [online]. *RFC2724.* [September 19, 2017]. Available from: https://tools.ietf.org/html/rfc2724

- Tanash, R., Chen, Z., Wallach, D. & Marschall, M. (2017). The Decline of Social Media Censorship and the Rise of Self-Censorship after the 2016 Failed Turkish Coup, in: *7th {USENIX} Workshop on Free and Open Communications on the Internet ({FOCI} 17)*. Vancouver, BC: {USENIX} Association.

- Theiler, J., (2014). Transductive and matched-pair machine learning for difficult target detection problems, in: *Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery* (Vol. 9088, p. 90880E). International Society for Optics and Photonics.

- Thomas, K., Grier, C., Ma, J., Paxson, V. & Song, D. (2011). Design and Evaluation of a Real-Time URL Spam Filtering Service, in: *Proceedings - IEEE Symposium on Security and Privacy*, (pp. 447–462).

- Thornton, C., Hutter, F., Hoos, H.H. and Leyton-Brown, K., (2013). Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms, in: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 847-855). ACM.

- Townsley, W., Valencia, A., Rubens, A., Pall, G., Zorn, G. & Palter, B. (1999). Layer Two Tunneling Protocol L2TP Status [online]. *RFC2661*. [January 11, 2018]. Available from: http://www.rfc-editor.org/rfc/rfc2661.txt

- Tselykh, A. & Petukhov, D. (2015). Web Service for Detecting Credit Card Fraud in near Real-Time, in: *Proceedings of the 8th International Conference on Security of Information and Networks - SIN '15*. (pp. 114–117). New York, New York, USA: ACM Press.

- Wade, J. (2010). A Biologically Inspired Training Algorithm for Spiking Neural Networks. *PhD thesis, University of Ulster*

- Wang, P., González, M., Menezes, R., Barabási, A.L. (2013) Understanding the spread of malicious mobile-phone programs and their damage potential. International journal of information security, Vol. 12, No. 5, pp: 383-392

- Werbos, P. (1975). Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. *PhD thesis, Harvard University*, (April).

- Wong, W., Giraldi, M., Magalhães, M. F. & Kangasharju, J. (2011). Content Routers: Fetching Data on Network Path. in: *IEEE International Conference on Communications (ICC 2011).* (pp. 1–6). IEEE.

- Wood, D., Stoss, V., Chan-Lizardo, L., Papacostas, G. S. & Stinson, M. E. (1988). Virtual Private Networks. in: *1988 International Conference on Private Switching Systems and Networks (ICPSSN 1988).* (pp. 132–136).

- Wurzinger, P., Platzer, C., Ludl, C., Kirda, E. & Kruegel, C. (2009). SWAP: Mitigating XSS Attacks Using a Reverse Proxy. in: *Proceedings of the 2009 ICSE Workshop on Software Engineering for Secure Systems, SESS 2009*, (pp. 33–39). IEEE Computer Society.

- Xiang, C., Yong, P. C. & Meng, L. S. (2008). Design of Multiple-Level Hybrid Classifier for Intrusion Detection System Using Bayesian Clustering and Decision Trees. *Pattern Recognition Letters*. 29(7). 918–924.

- Yang, M., Luo, J., Ling, Z., Fu, X. & Yu, W. (2015). De-Anonymizing and Countermeasures in Anonymous Communication Networks. *IEEE Communications Magazine*. 53(4). 60–66.

- Yu, F. Y. F., Chen, Z. C. Z., Diao, Y. D. Y., Lakshman, T. V. & Katz, R. H. (2006). Fast and Memory-Efficient Regular Expression Matching for Deep Packet Inspection. *2006 Symposium on Architecture For Networking And Communications Systems*, 1–10.

- Zorn, G., Pall, G. S., Hamzeh, K., Verthein, W., Taarud, J. & Litte, W. (1999). Point-to-Point Tunneling Protocol (PPTP) [online]. *RFC 2637*. [January 12, 2018]. Available from: https://tools.ietf.org/html/rfc2637

Appendix A – Packet Capture Script

Packet capture script for chapter 3

```
# Packetcap.py - Script to capture HTTP and HTTPS packets and log them
# to a .CSV file.
import socket
import sys
import csv
from struct import *
# Create an INET, STREAMing socket
try:
    s = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_TCP)
except socket.error:
    print('Socket could not be created.')
    sys.exit()
outputFile = open('vpntraffictest.csv', 'w', newline='')
writer = csv.writer(outputFile)
# Write out the top row
writer.writerow(['Version', 'Protocol', 'TTL', 'SrcAddr', 'DestAddr',
                 'SrcPort', 'DestPort', 'SeqNum', 'AckNum', 'Flag', 'dataSize',
                 'Service', 'Label'])
# receive a packet
while True:
    packet = s.recvfrom(65565)
    # Transfer tuple contents to string type.
    packet = packet[0]
    # Take first 20 bytes for the ip header.
    # Ethernet header is usually before, but we aren't capturing that.
    ip_header = packet[0:20]
    # Unpack from bytes format
    iph = unpack('!BBHHHBBH4s4s', ip_header)
    version_ihl = iph[0]
    version = version_ihl >> 4
    ihl = version_ihl & 0xF
    iph_length = ihl * 4
    ttl = iph[5]
    protocol = iph[6]
    s_addr = socket.inet_ntoa(iph[8])
    d_addr = socket.inet_ntoa(iph[9])
    # TCP header starts right after IP header and is usually
    # 20 bytes long
    tcp_header = packet[20:40]
```

```
    # Unpack from bytes format
    tcph = unpack('!HHLLBBHHH', tcp_header)
    source_port = tcph[0]
    dest_port = tcph[1]
    sequence = tcph[2]
    acknowledgement = tcph[3]
    doff_reserved = tcph[4]
    tcph_length = doff_reserved >> 4
    h_size = iph_length + tcph_length * 4
    data_size = len(packet) - h_size
    # Select bytes containing tcp flags and label them
    tcpFlag = packet[33:34].hex()
    if tcpFlag == "01":
       Flag = "FIN"
    elif tcpFlag == "02":
       Flag = "SYN"
    elif tcpFlag == "03":
       Flag = "FIN-SYN"
    elif tcpFlag == "08":
       Flag = "PSH"
    elif tcpFlag == "09":
       Flag = "FIN-PSH"
    elif tcpFlag == "0A":
       Flag = "SYN-PSH"
    elif tcpFlag == "10":
       Flag = "ACK"
    elif tcpFlag == "11":
       Flag = "FIN-ACK"
    elif tcpFlag == "12":
       Flag = "SYN-ACK"
    elif tcpFlag == "18":
       Flag = "PSH-ACK"
    else:
       Flag = "OTH"
    # If statement to select only HTTP and HTTPS packets for
    # logging
    if source_port == 80 or source_port == 443:
       if source_port == 80:
          writer.writerow([str(version), str(protocol),
                           str(ttl), str(s_addr),
                           str(d_addr), str(source_port),
                           str(dest_port), str(sequence),
                           str(acknowledgement), Flag,
                           str(data_size), "HTTP", "0"])
```

```
            print("Packet Captured")
        else:
            writer.writerow([str(version), str(protocol),
                             str(ttl), str(s_addr),
                             str(d_addr), str(source_port),
                             str(dest_port), str(sequence),
                             str(acknowledgement), Flag,
                             str(data_size), "HTTPS", "0"])
            print("Packet Captured")
outputFile.close()
```

Automatic browsing script for chapter 3

```
import time
from splinter import Browser

# Create instance of Browser object
b = Browser()
url = ["http://whatismyipaddress.com/"]
# List of glype proxy sites to be visited
# "http://www.blackhost.xyz/glype/", "http://proxy.lelouet.fr/",
url = [
      "https://secure.cogsoz.com/proxy/",
      "http://samstevenm.net/prox/", "https://muadness.com/proxy/",
      "http://www.radiocarb.com/p/", "http://proxy.rimmer.su/",
      "https://awssl.com/", "https://moka4.com/",
      "https://webproxy.stealthy.co/", "http://bvpn.win/", "http://www.emuby.com/",
      "http://www.docoja.com/blue/index.php"]
for site in url:
   # Visit the site using Browser
   b.visit(site)
   time.sleep(2)
   # Find and fill the textbox then find the submit button and 'click' it
   b.find_by_id('input').fill('www.whatismyipaddress.com')
   time.sleep(1)
   if b.is_element_present_by_css('input.button'):
      goButton = b.find_by_css('input.button')
      goButton.click()
   elif b.is_element_present_by_css('input.submitbutton'):
      goButton = b.find_by_css('input.submitbutton')
      goButton.click()
   time.sleep(2)
```

```
    # Deal with SSL warning page if it appears
    sslWarningPage = b.find_by_text('Warning!')
    if sslWarningPage is not None:
        print('Warning encountered, dealing with it...')

        continueButton = b.find_by_css('input')[1]
        continueButton.click()

b.quit
```